BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 18

Asymmetry & Concurrency Aware Storage Management

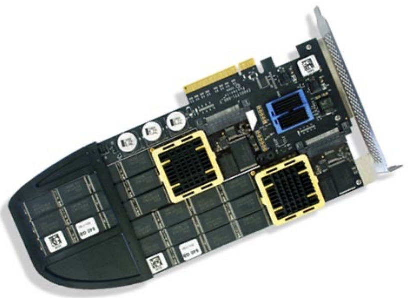
Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/

Evolution of Storage Hierarchy

HDD

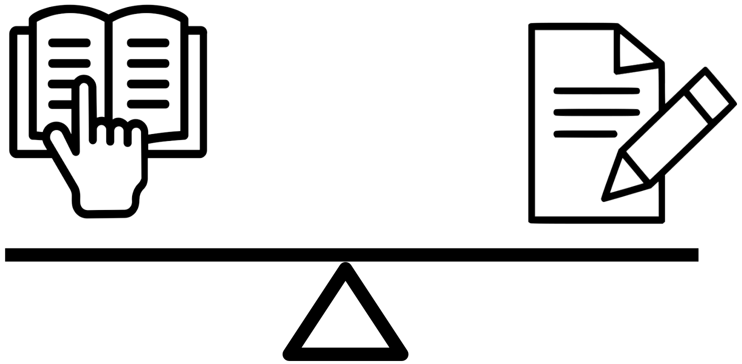
SSD



NVM SSD

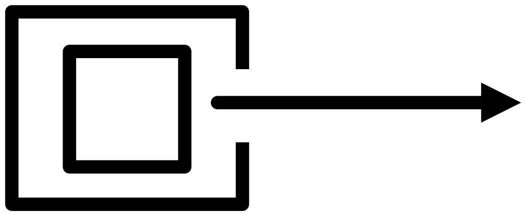
Hard Disk Drives

Hard Disk Drives



Symmetric cost for **Read & Write** to disk

✓



One I/O at a time

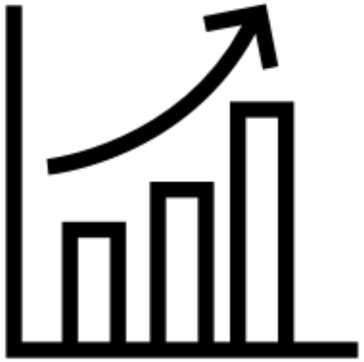
✓

Solid-State Drive (SSD)

BOSTON UNIVERSITY

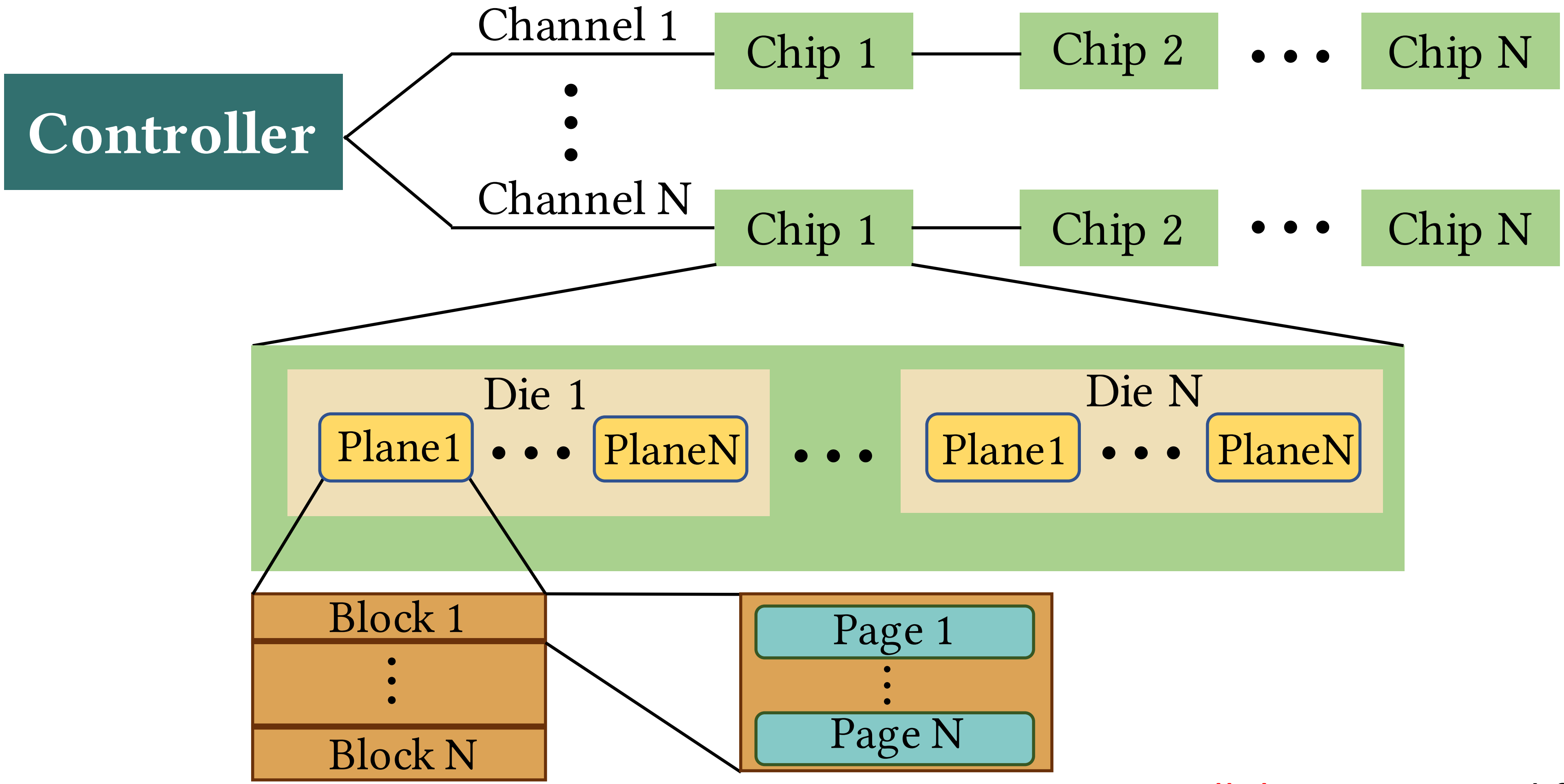
Solid-State Drive (SSD)

No mechanical movement



Fast access, High chip density, Low energy consumption

Internals of an SSD



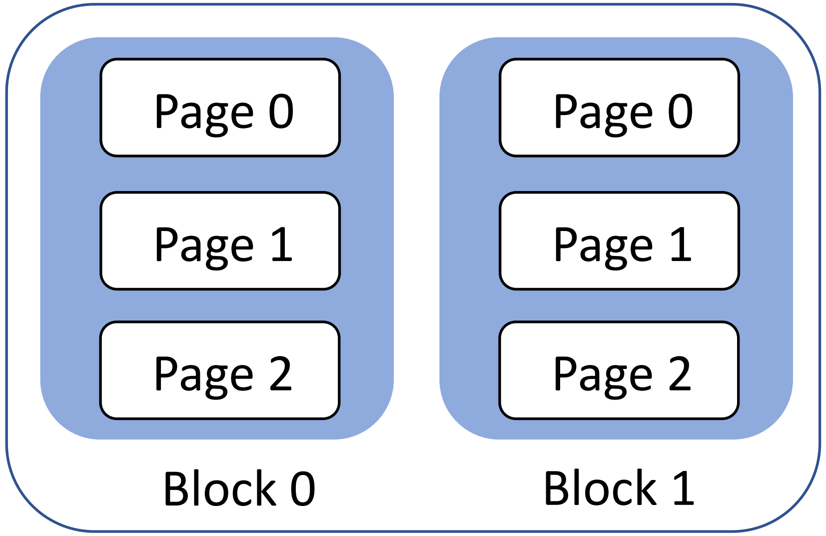
Parallelism at many different levels (channel, chip, die, plane, block)

BOSTON UNIVERSITY

Writes in SSD

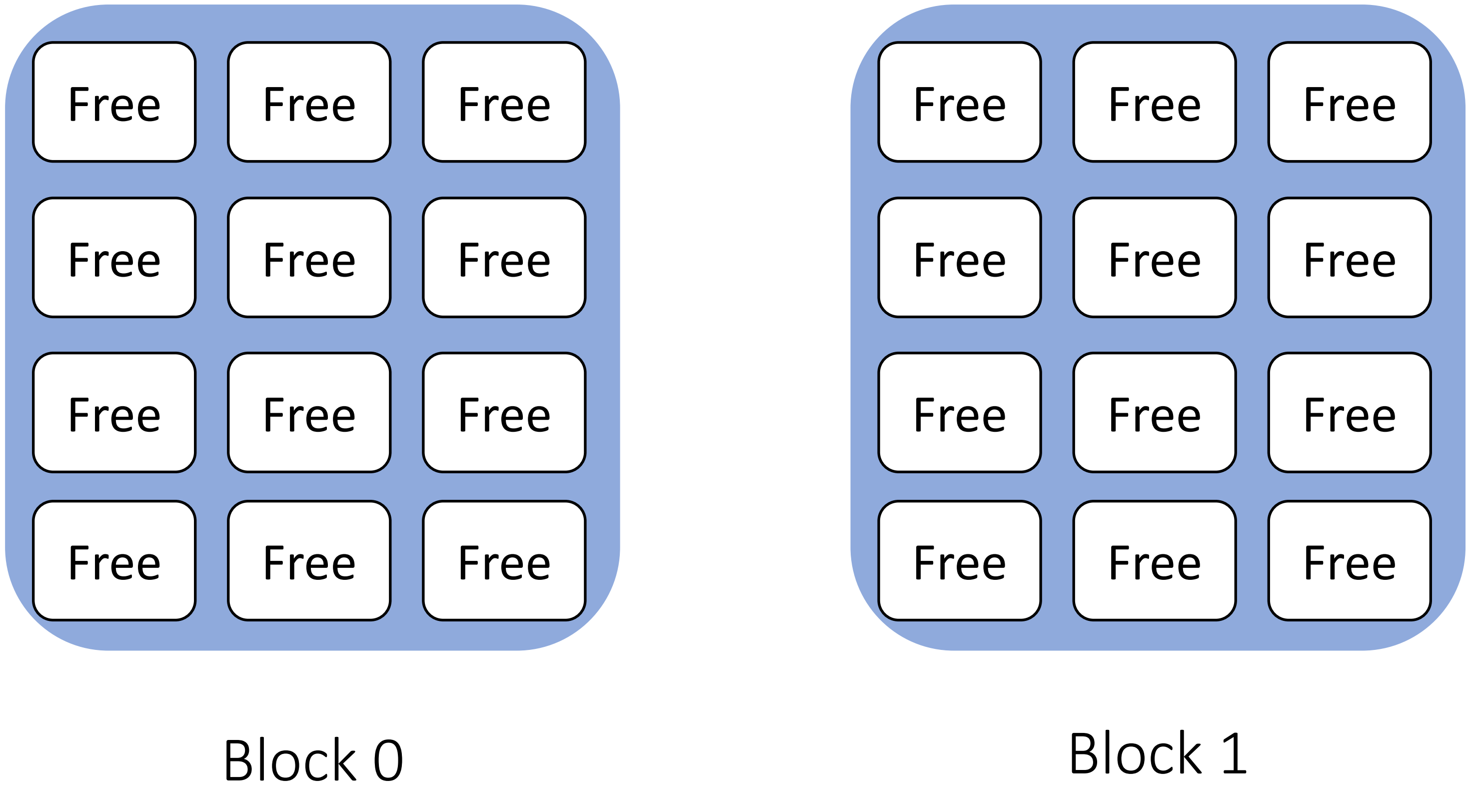
Out-of-place updates cause invalidation

Invalidation causes garbage collection.

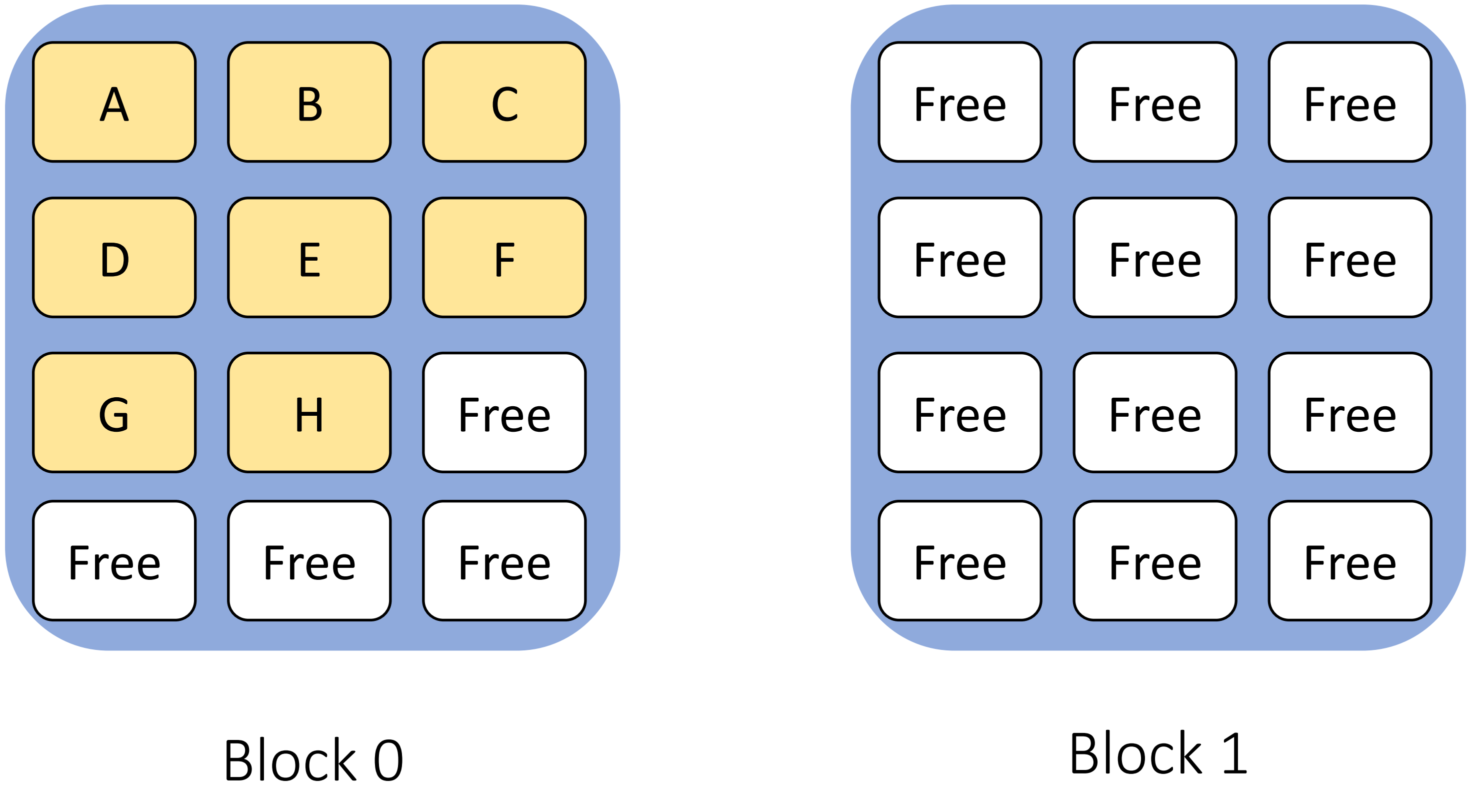


Plane

Writes in SSD



Writes in SSD



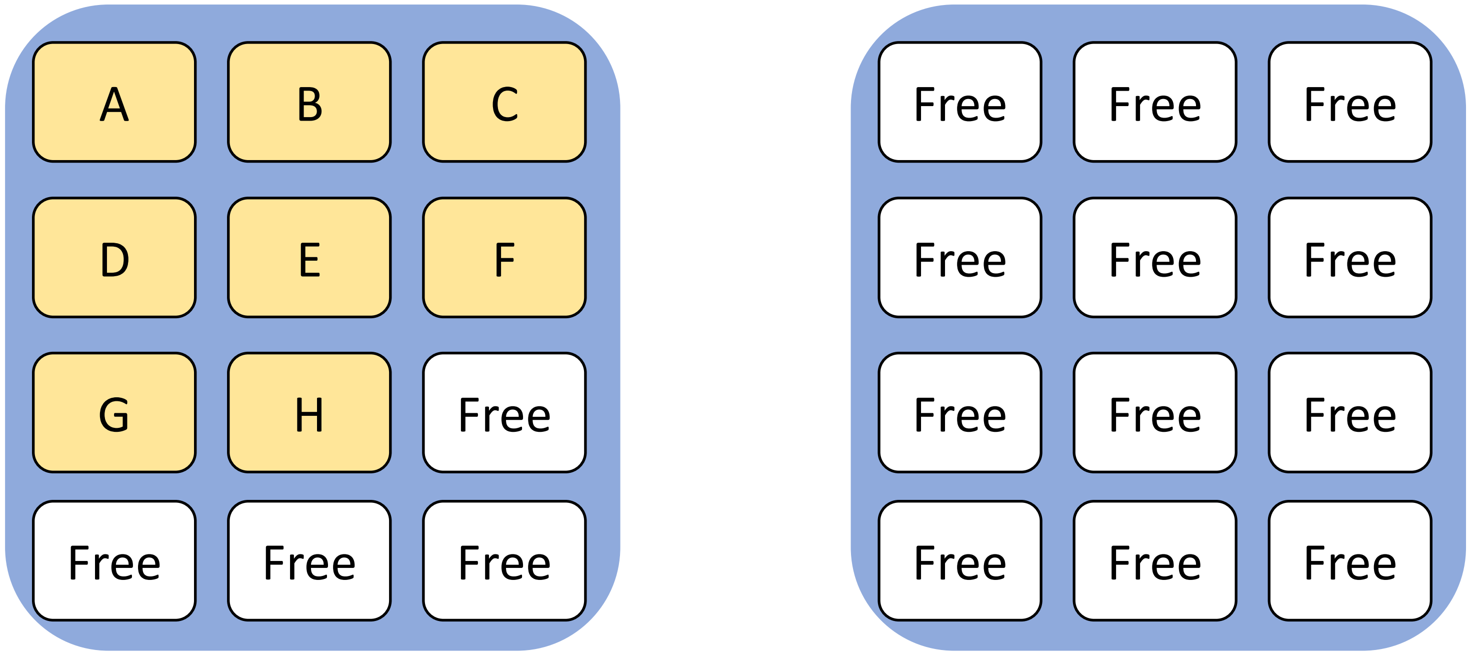
Writing in a free page isn't costly!

BOSTON UNIVERSITY

Writes in SSD

Update

A, B, C, D



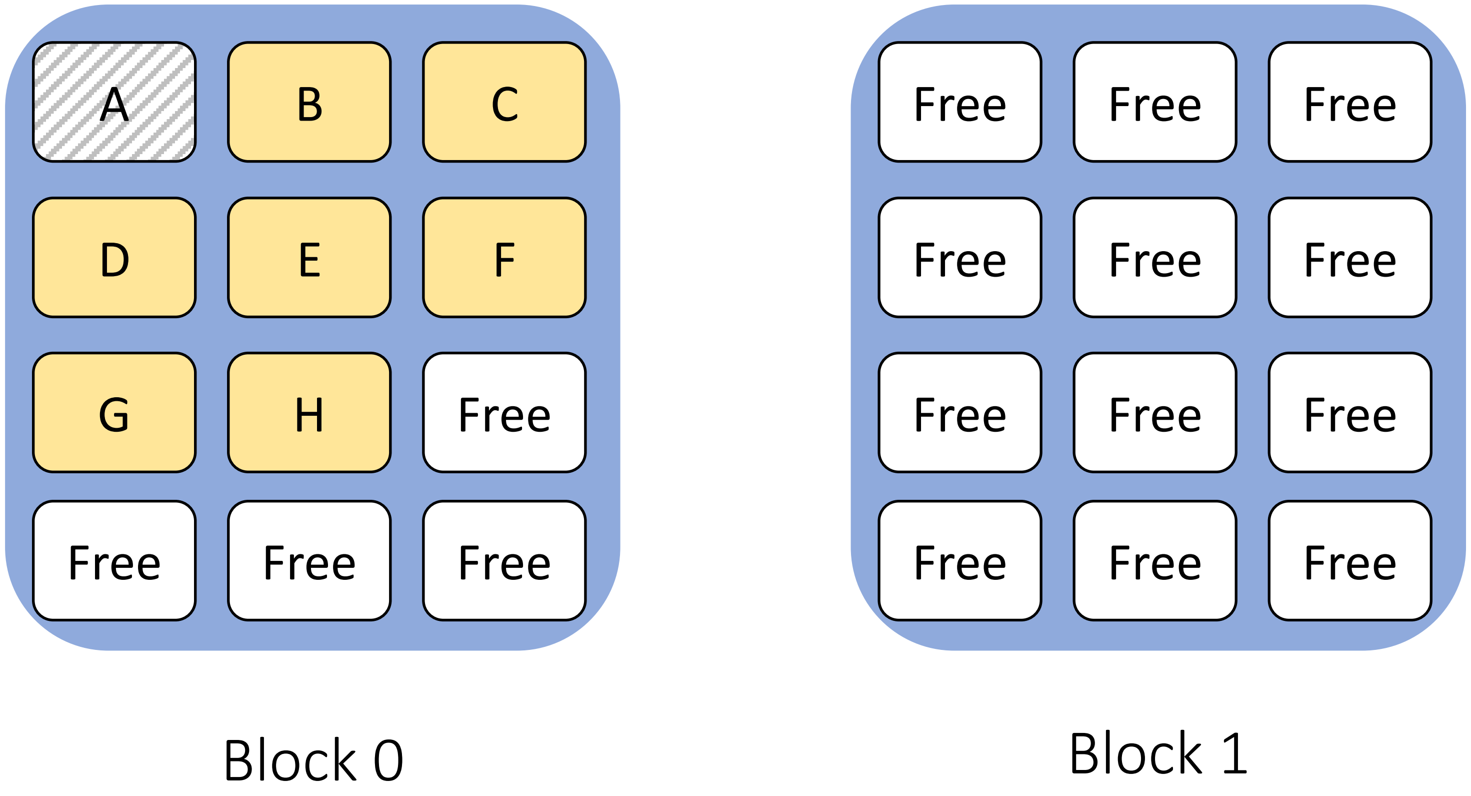
Block 0

Block 1

Writes in SSD

Update

A, B, C, D

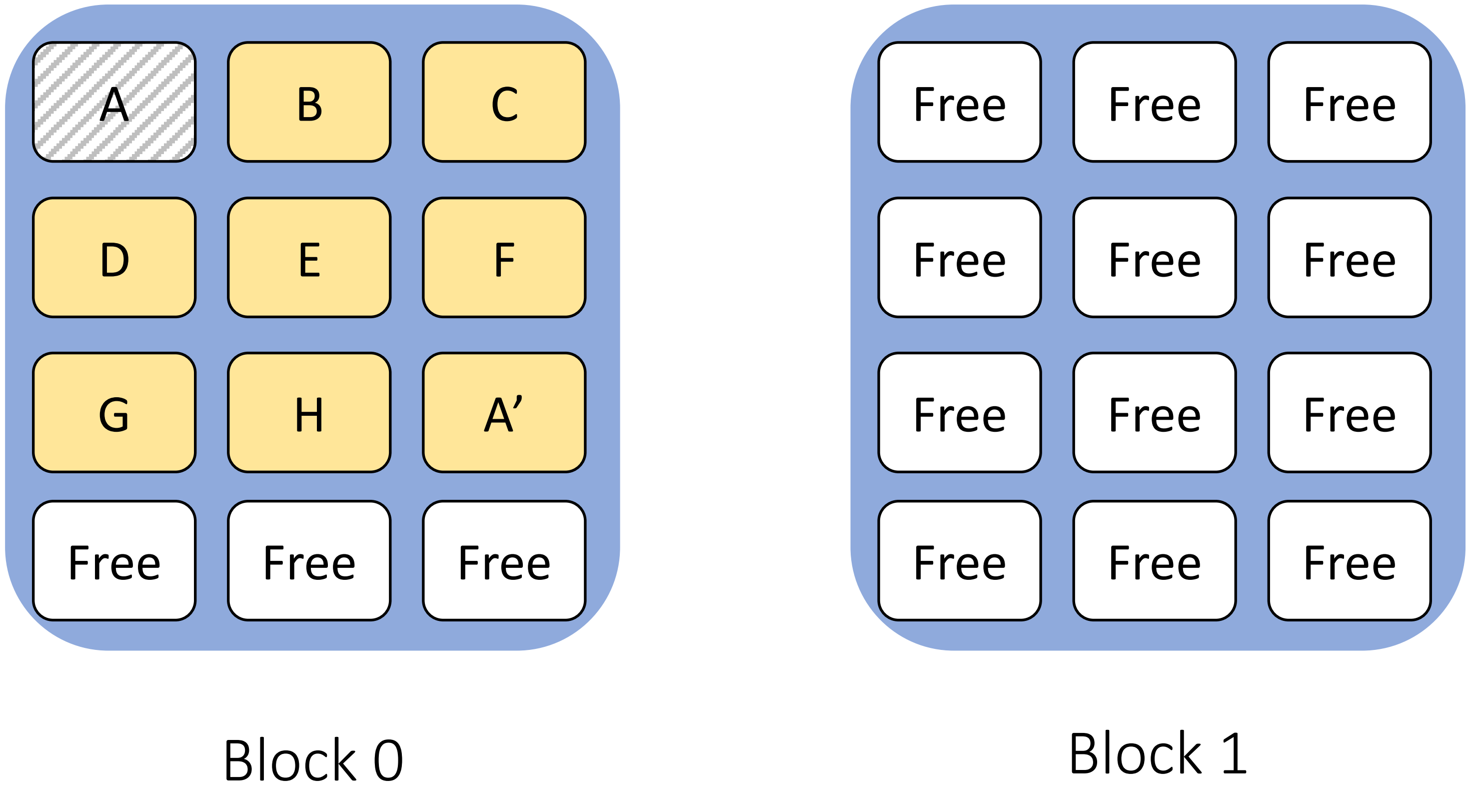


BOSTON UNIVERSITY

Writes in SSD

Update

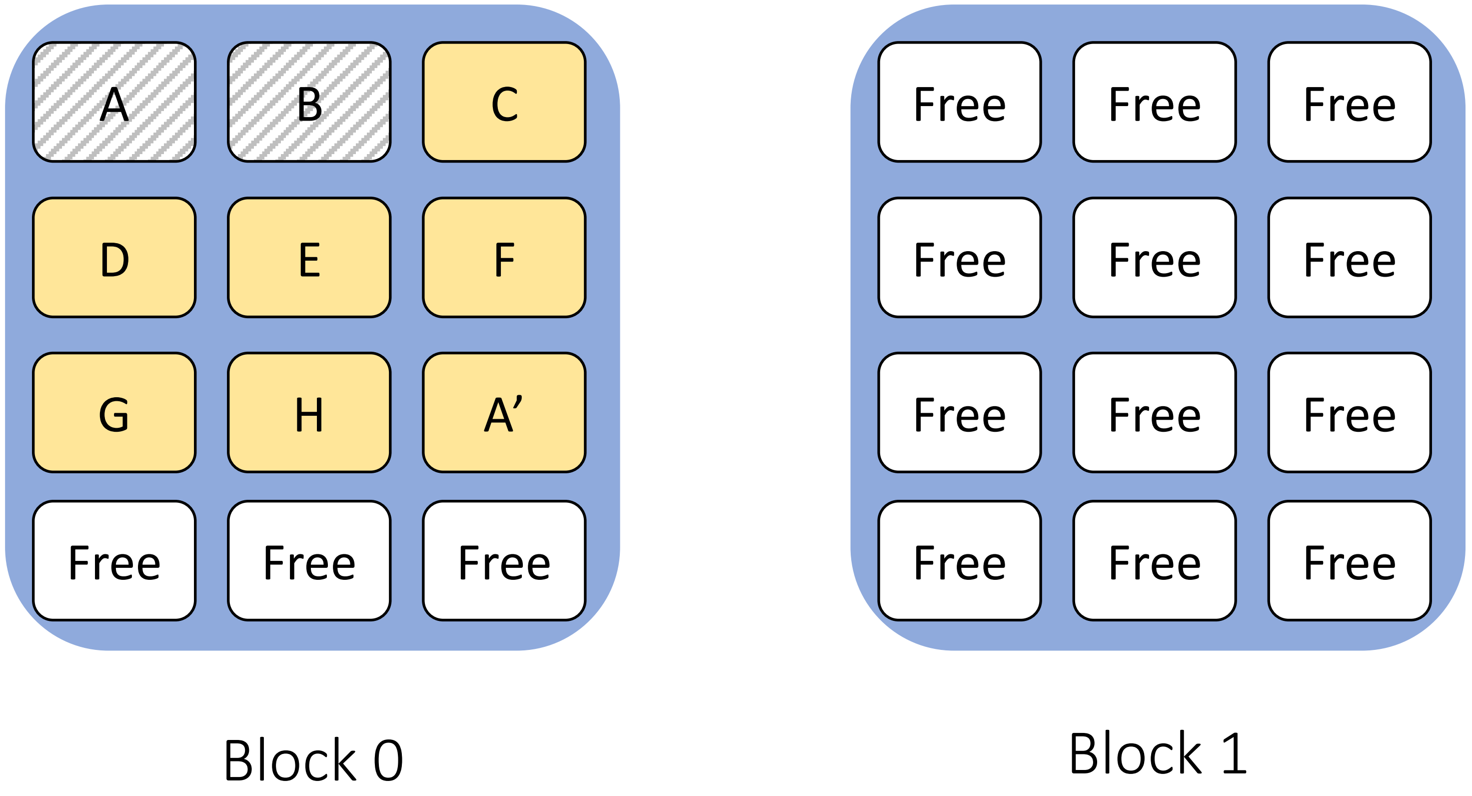
A, B, C, D



Writes in SSD

Update

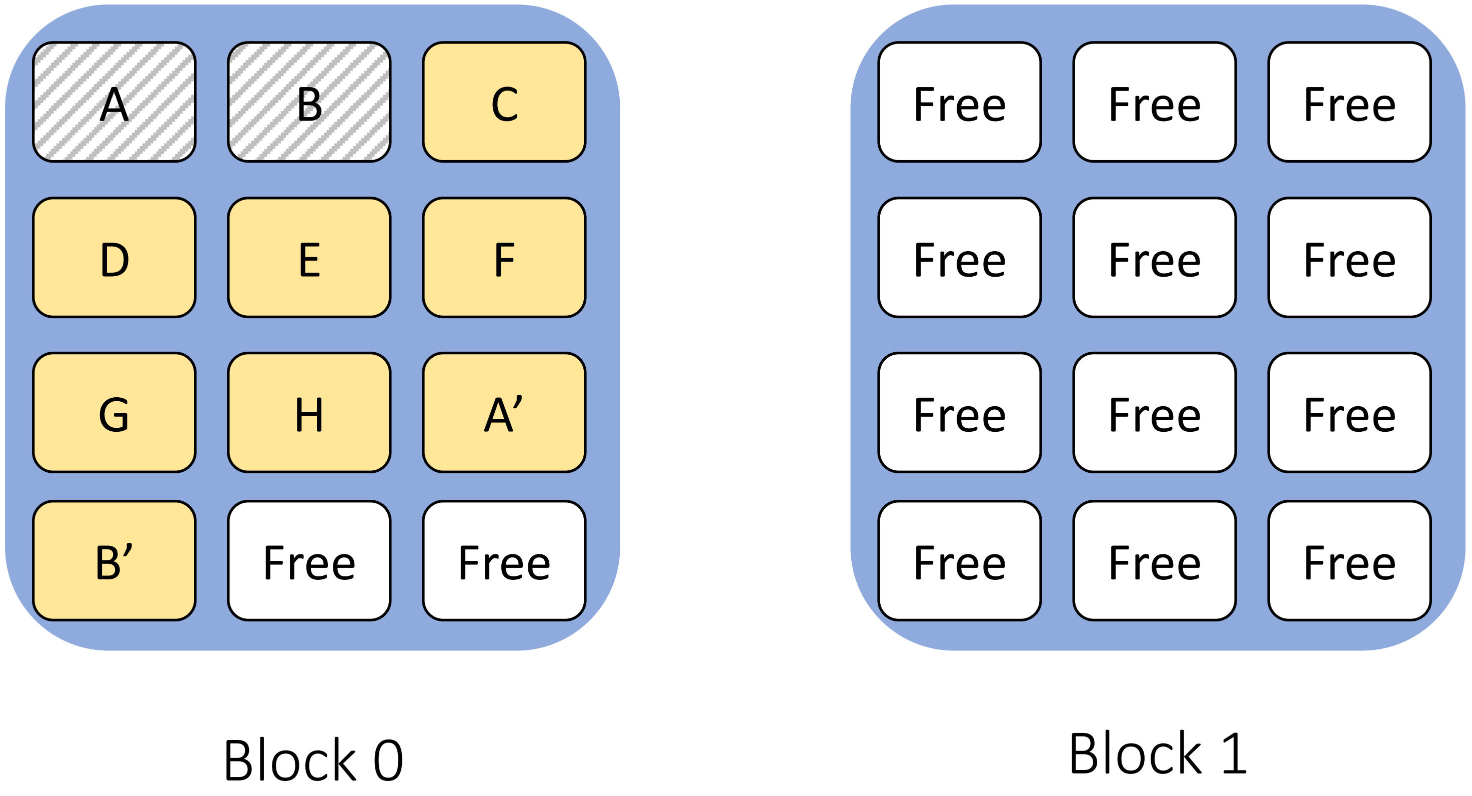
A, **B**, C, D



Writes in SSD

Update

A, **B**, C, D

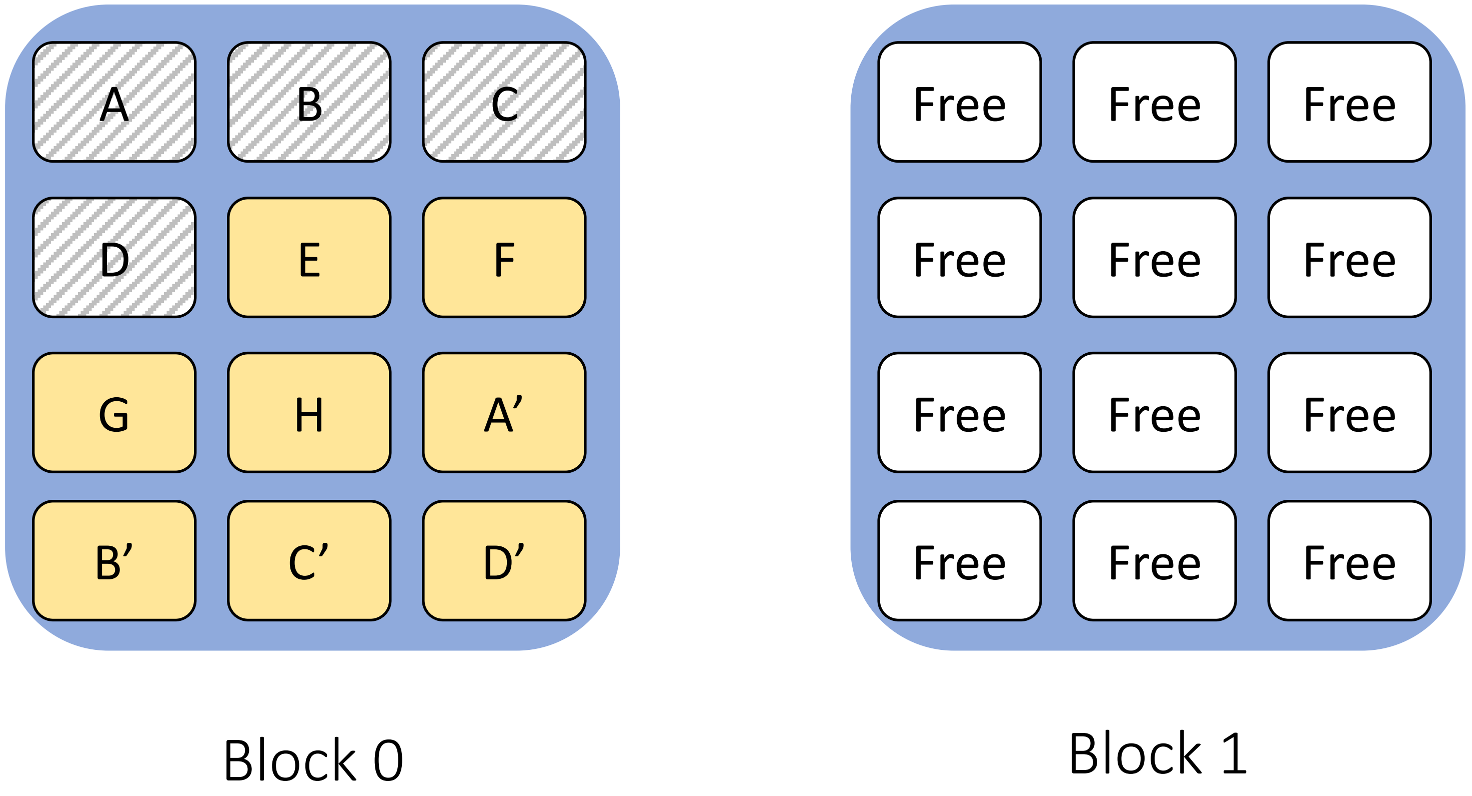


BOSTON UNIVERSITY

Writes in SSD

Update

A, B, C, D

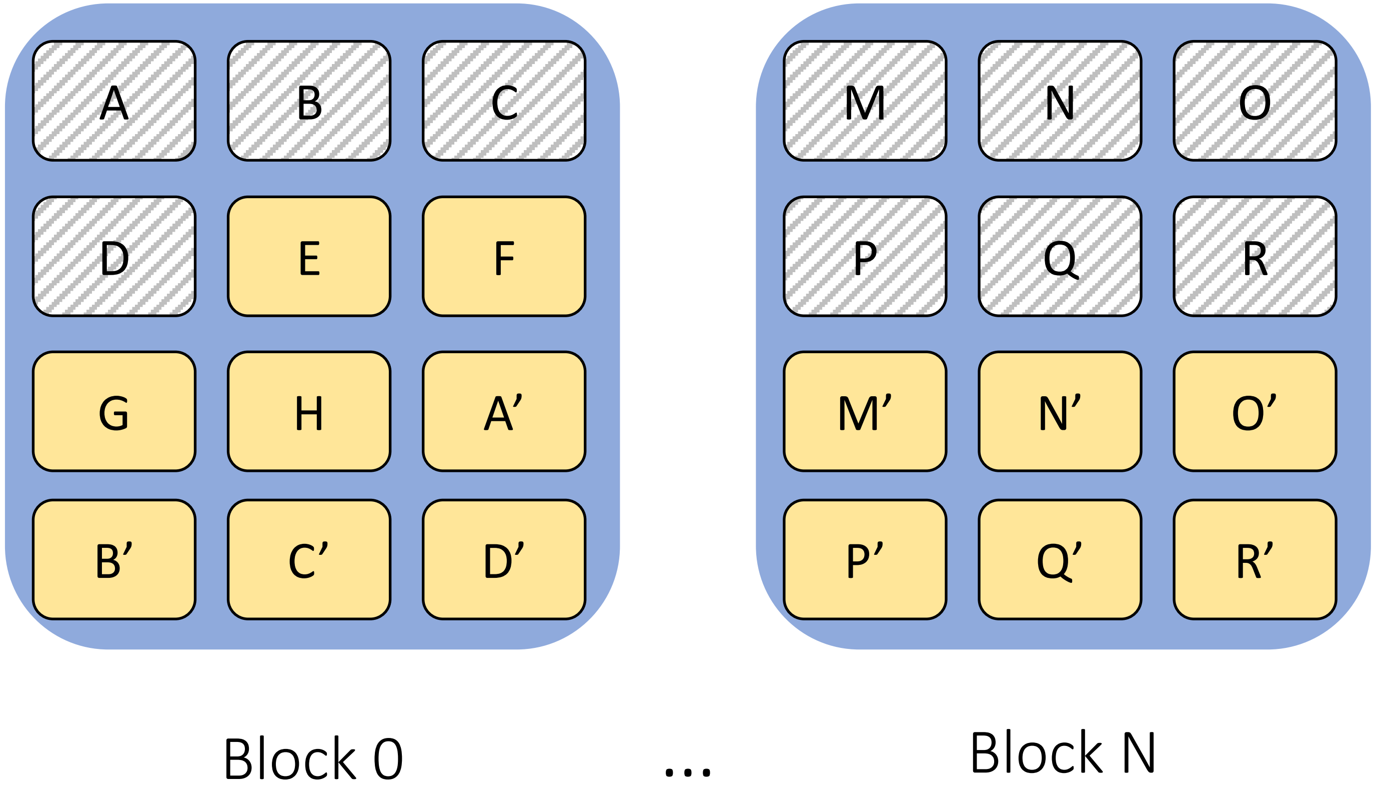


Not all updates are costly!

BOSTON UNIVERSITY

Writes in SSD

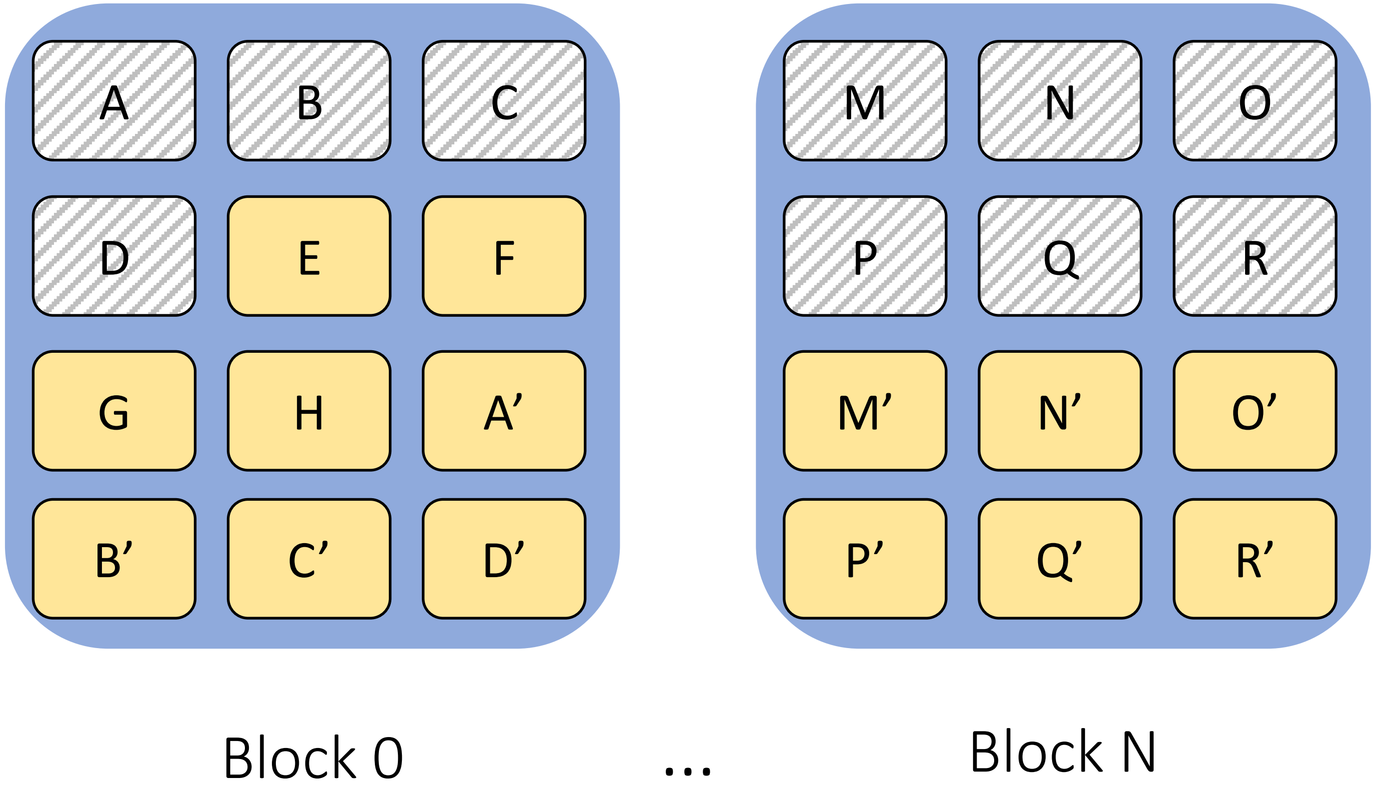
What if there is no space?



Writes in SSD

What if there is no space?

Garbage Collection!

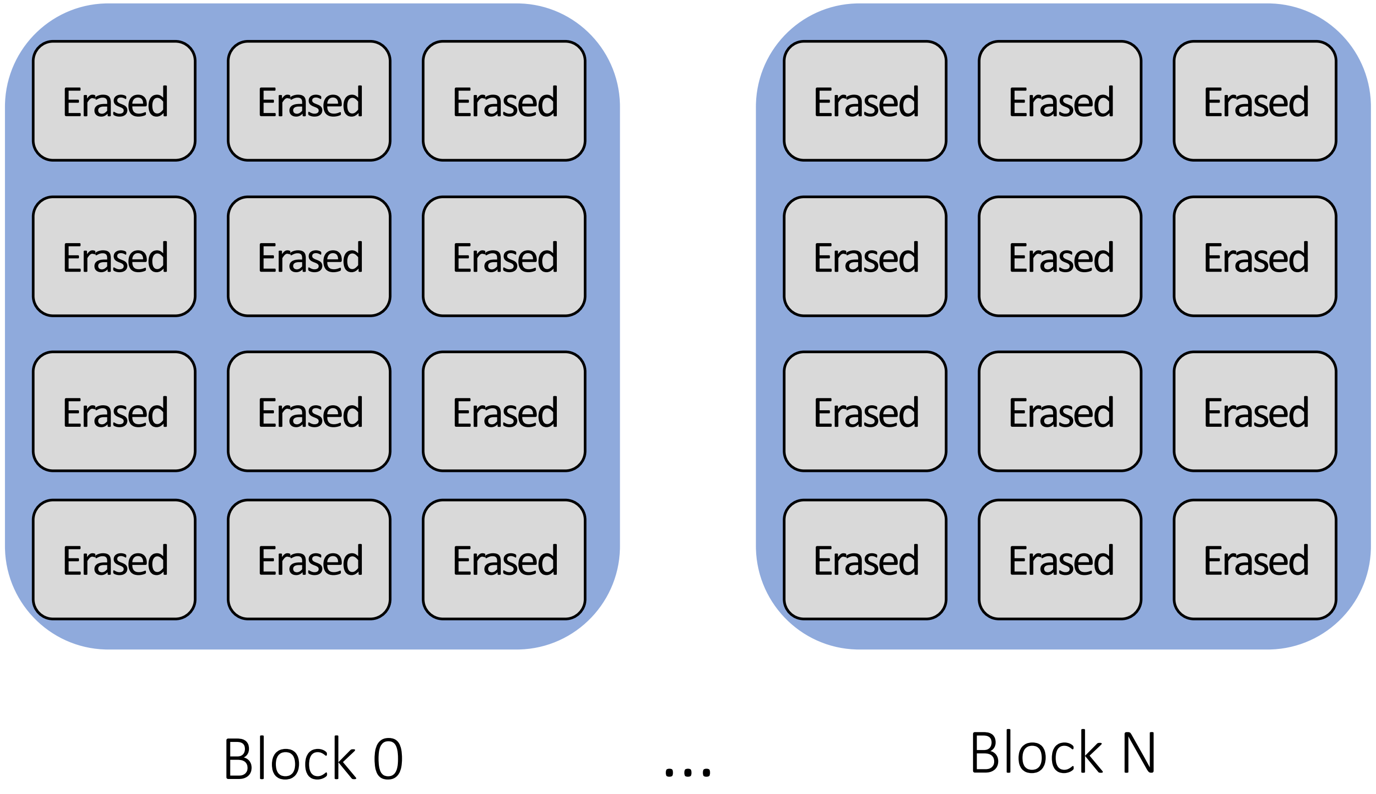


BOSTON UNIVERSITY

Writes in SSD

What if there is no space?

Garbage Collection!



Valid pages:

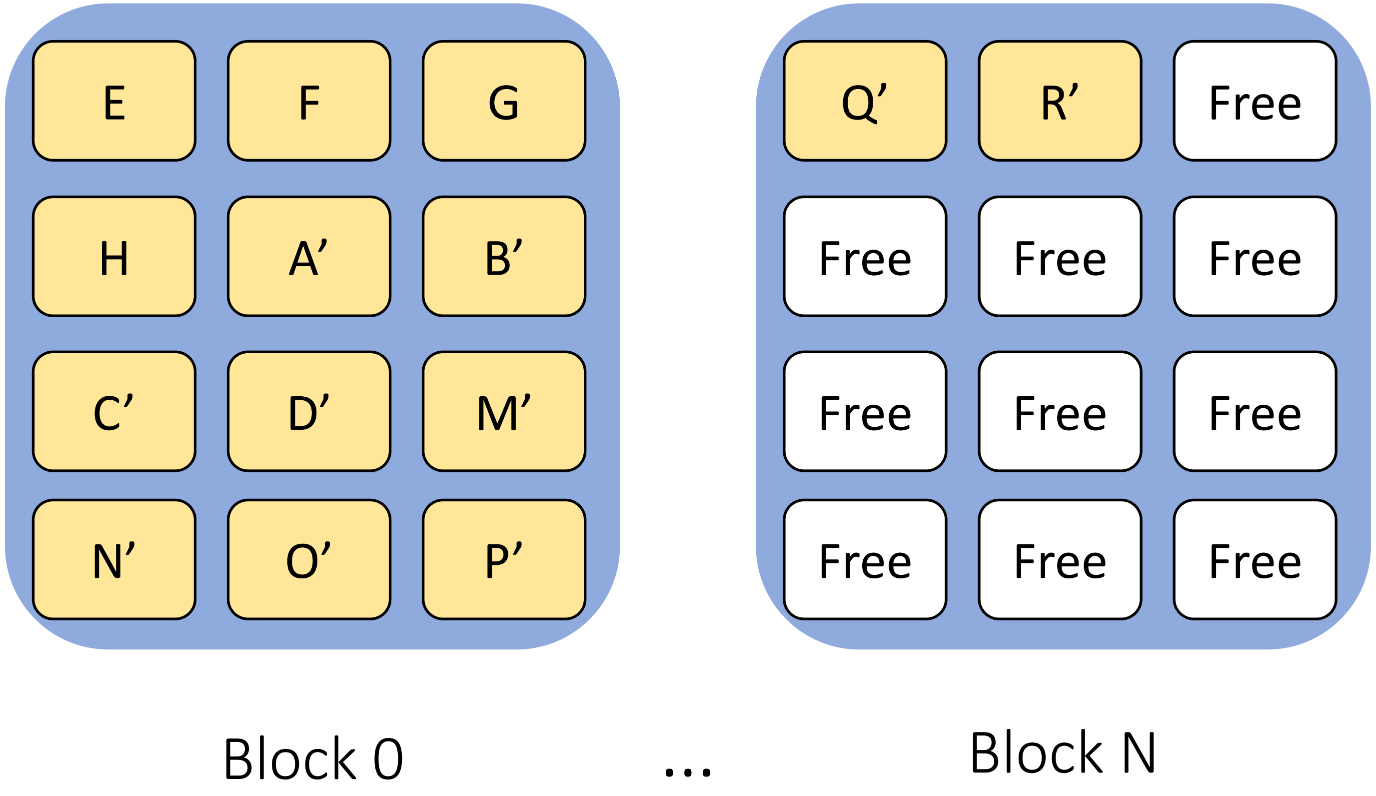
E	F	G	H	A’	B’	C’	D’	M’	N’	O’	P’	Q’	R’

BOSTON UNIVERSITY

Writes in SSD

What if there is no space?

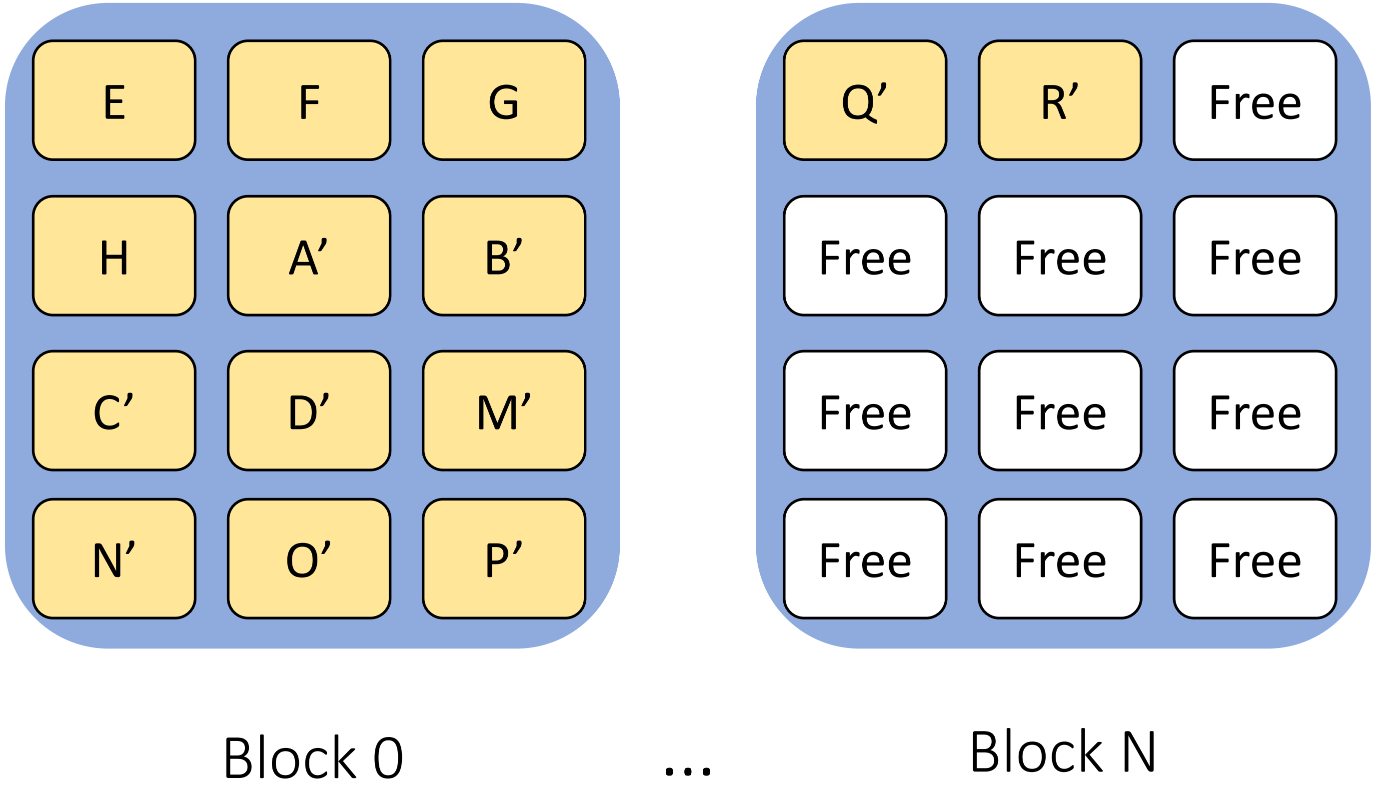
Garbage Collection!



Writes in SSD

What if there is no space?

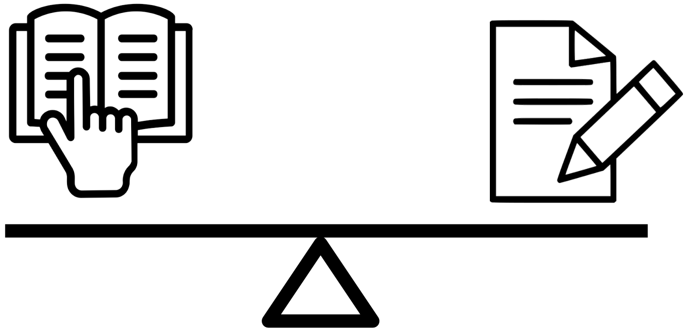
Garbage Collection!



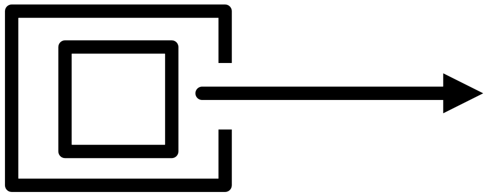
Higher average update cost (due to GC) → ***Read/Write asymmetry***

HDD vs SSD

HDD

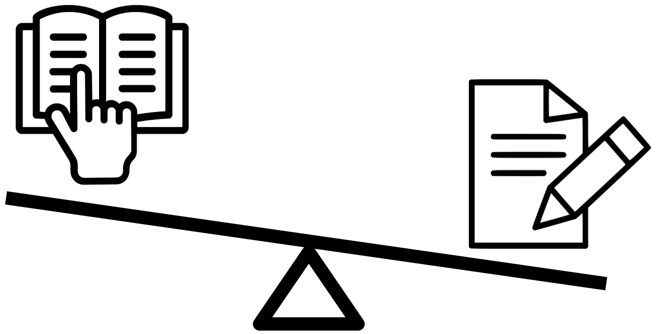


Symmetric cost for Read & Write

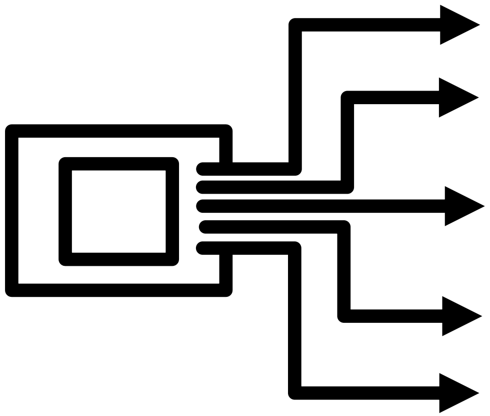


One I/O at a time

SSD



Read/Write Asymmetry



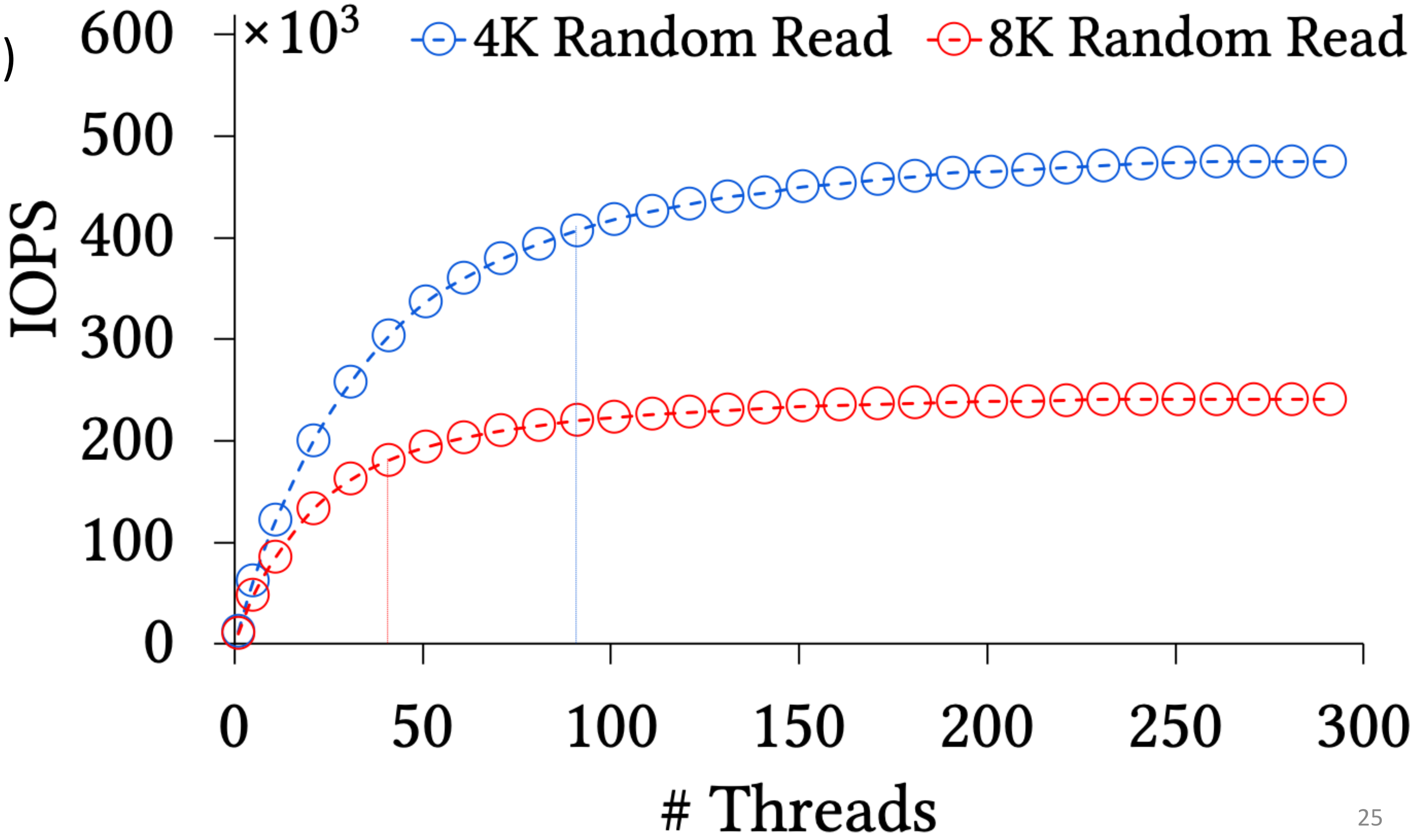
Concurrency

BOSTON UNIVERSITY

Measuring Asymmetry/Concurrency (With FS)

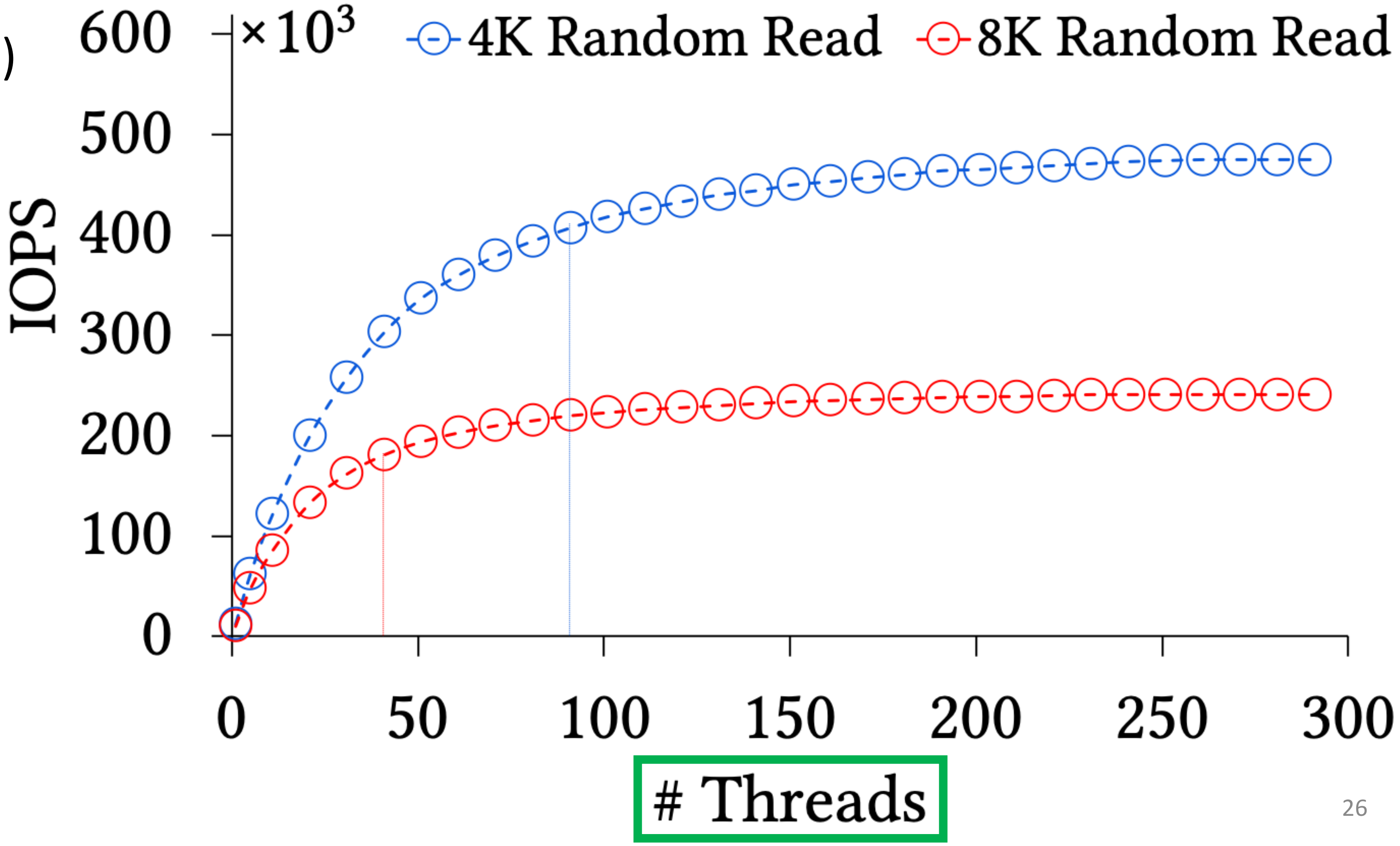
Device

PCIe SSD - P4510 (1TB)



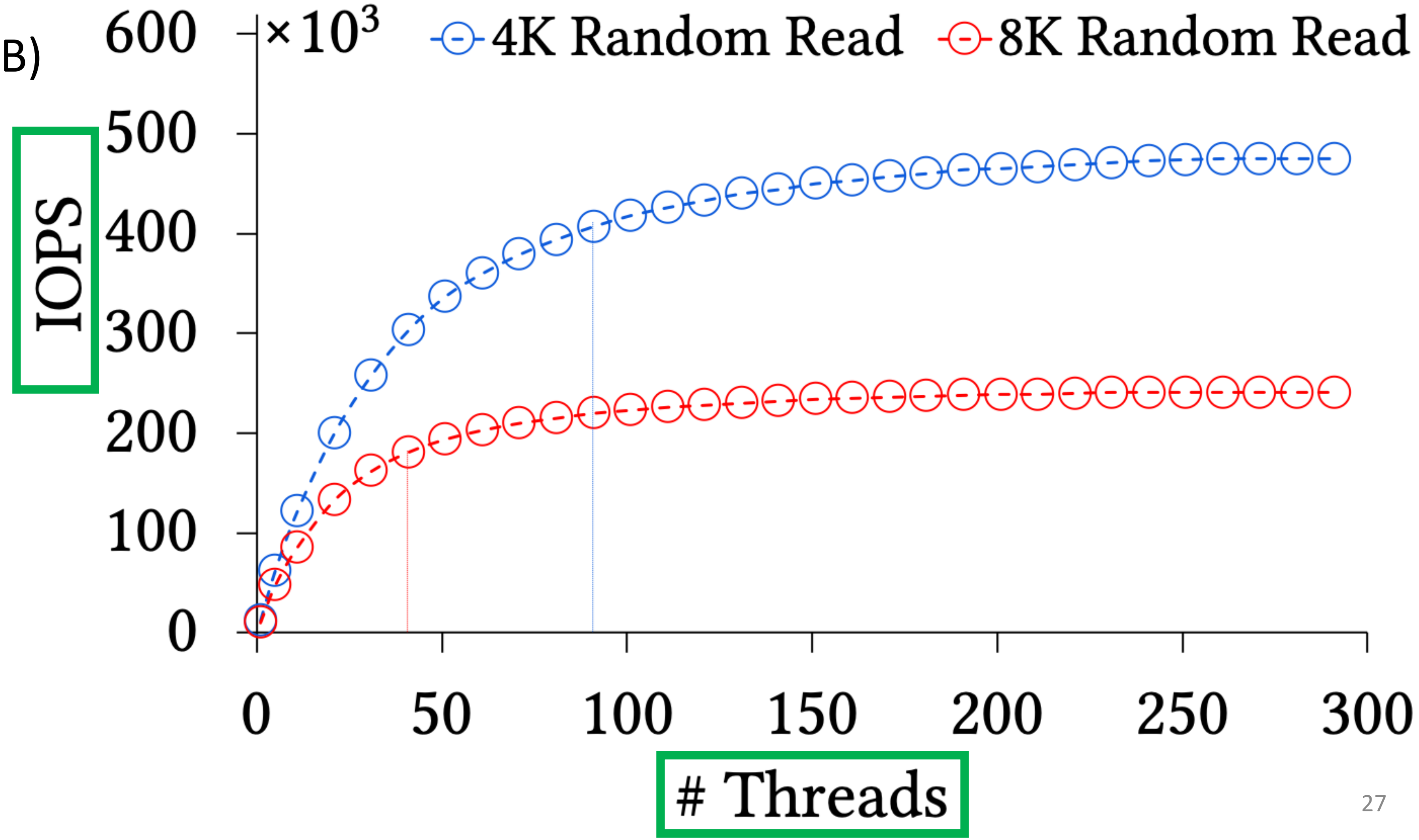
Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)



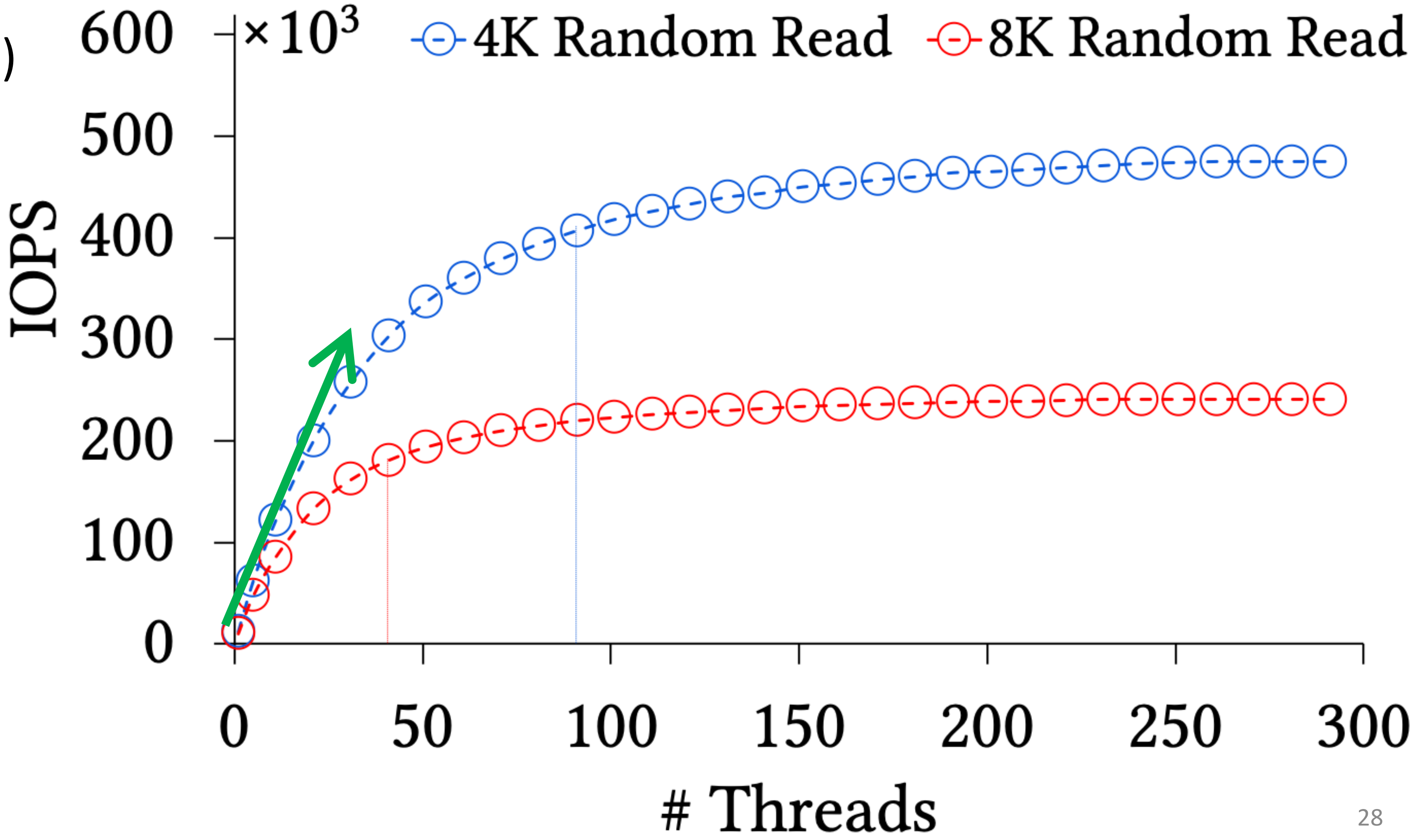
Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)



Measuring Asymmetry/Concurrency (With FS)

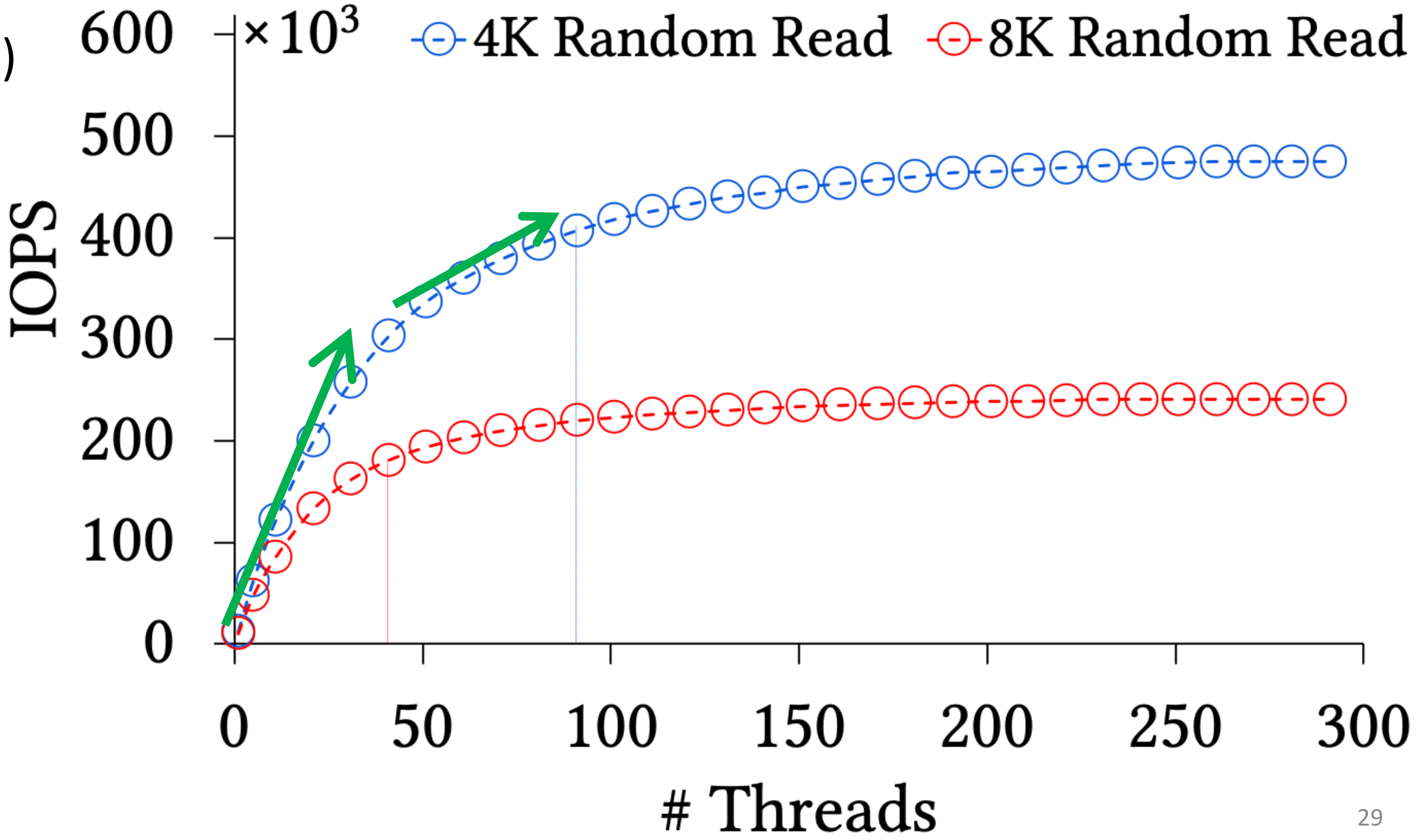
Device
PCIe SSD - P4510 (1TB)



BOSTON UNIVERSITY

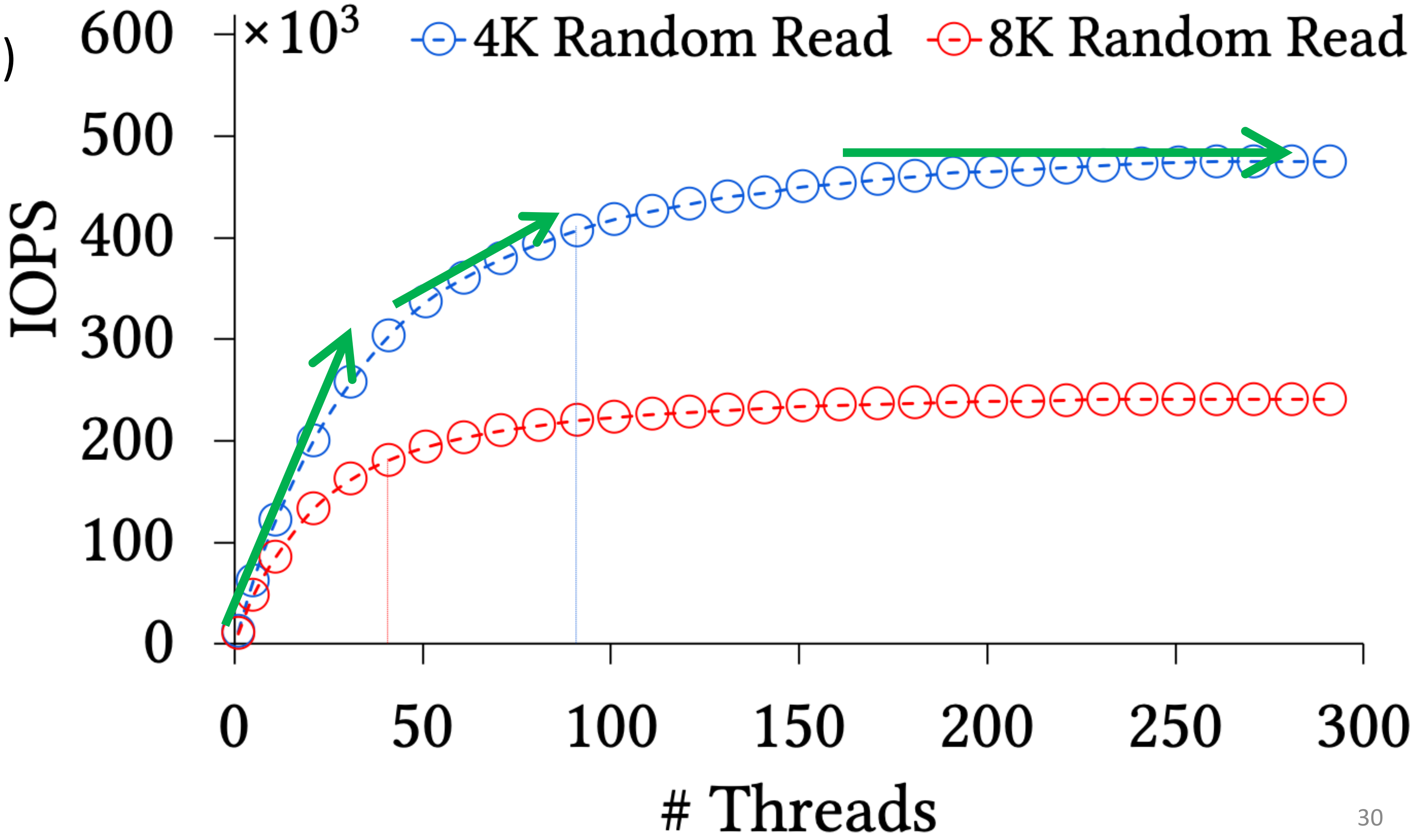
Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)



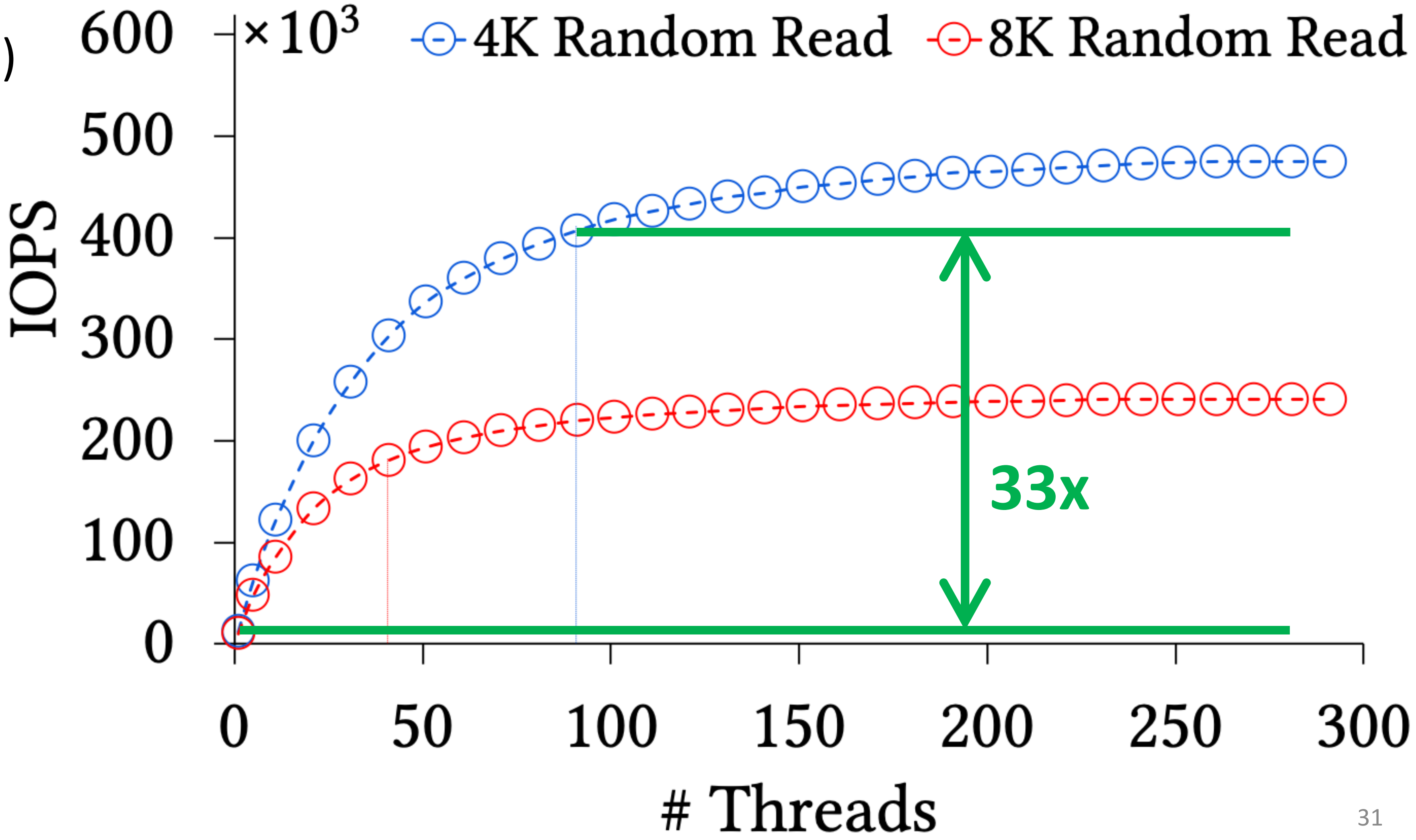
Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)



Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)

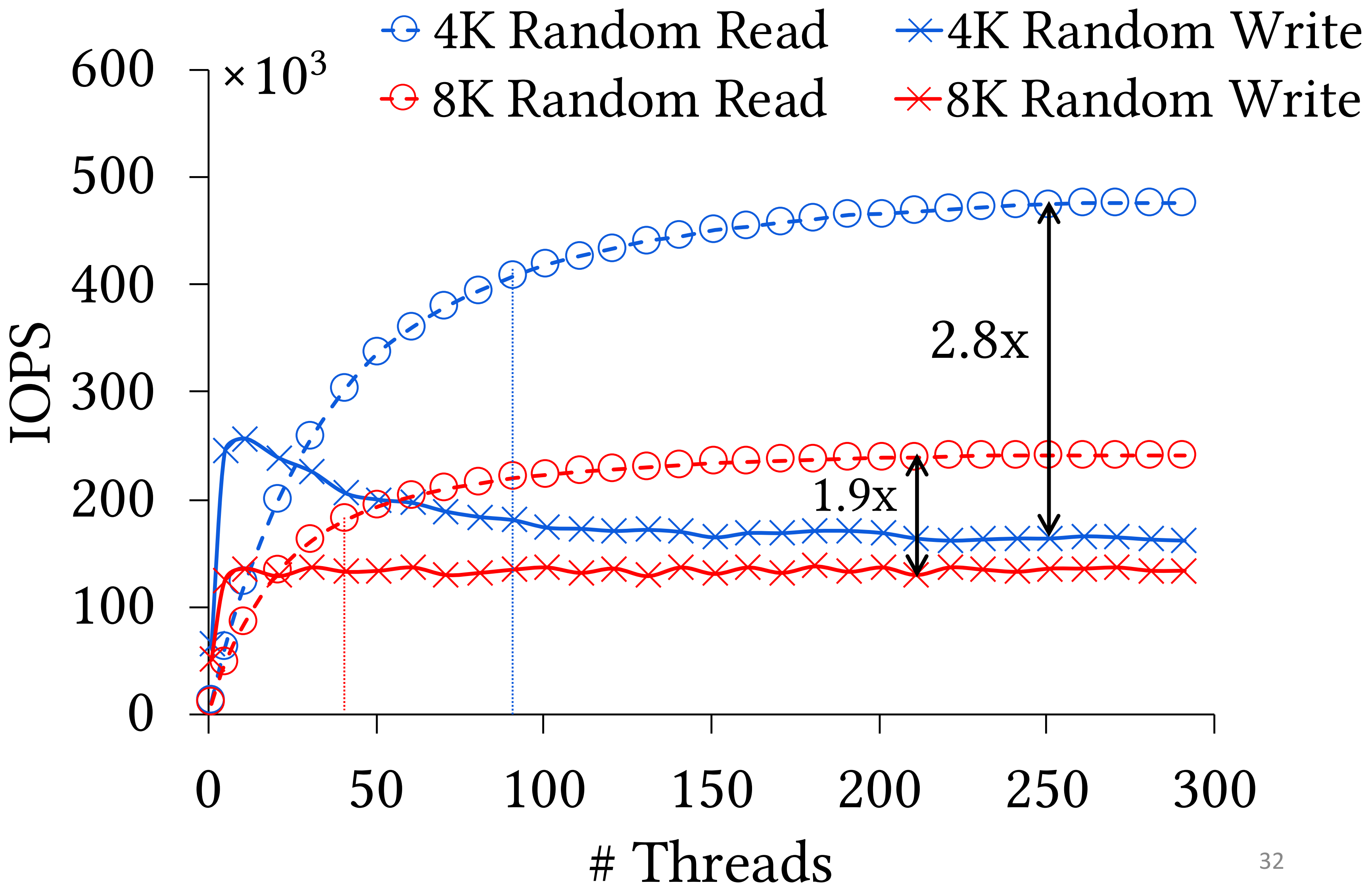


BOSTON UNIVERSITY

Measuring Asymmetry/Concurrency (With FS)

Device

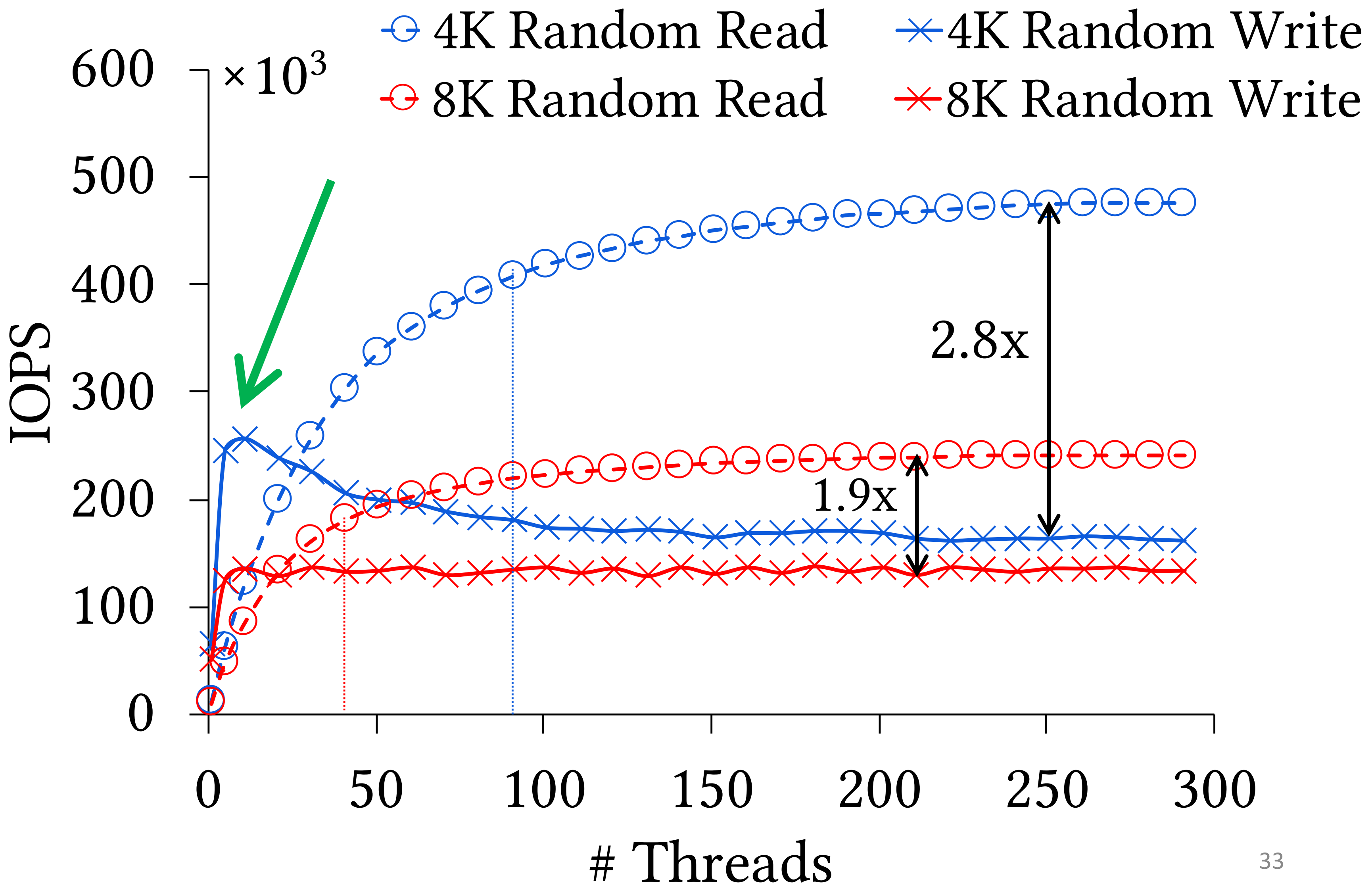
PCIe SSD - P4510 (1TB)



BOSTON UNIVERSITY

Measuring Asymmetry/Concurrency (With FS)

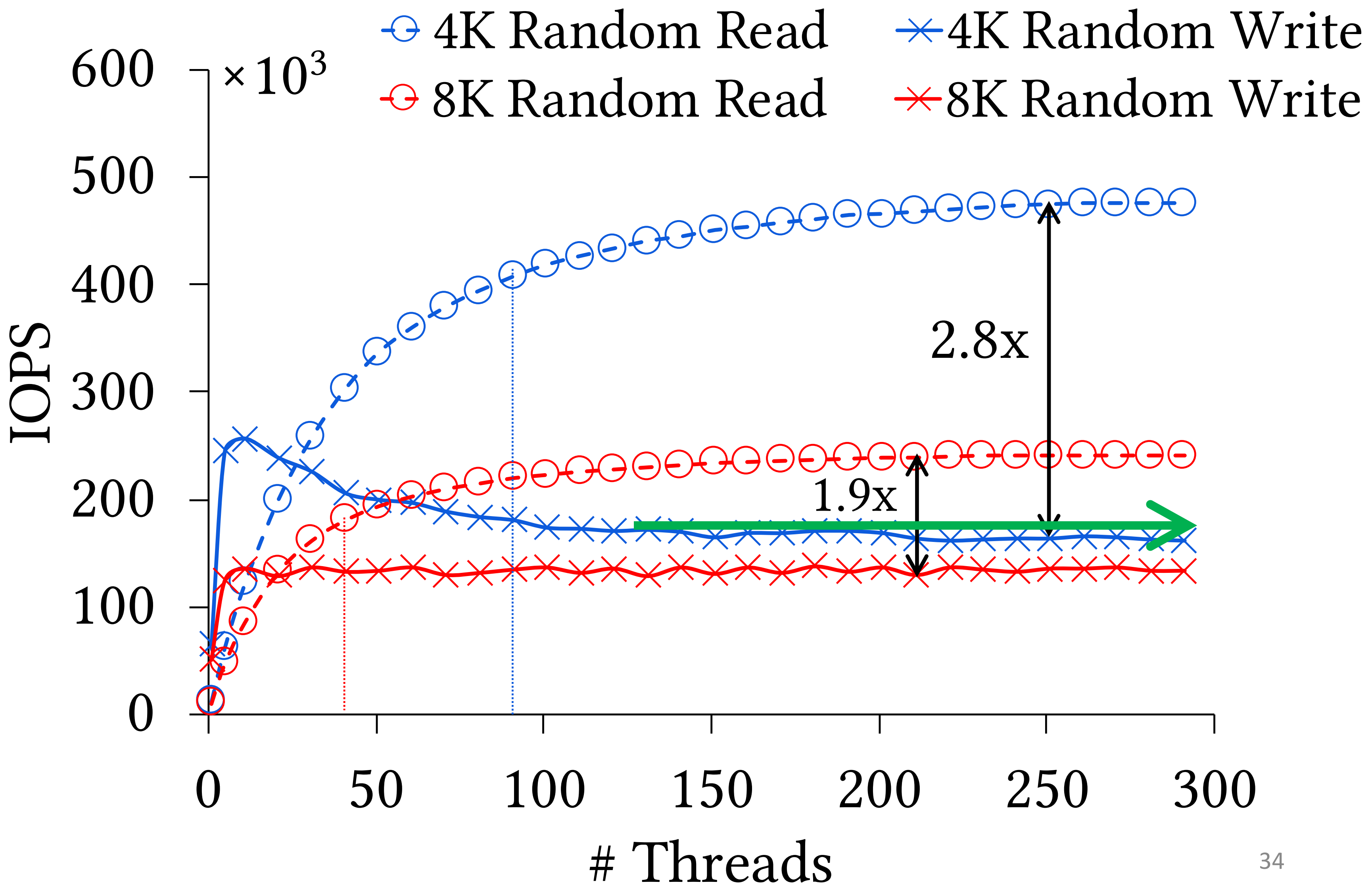
Device
PCIe SSD - P4510 (1TB)



Measuring Asymmetry/Concurrency (With FS)

Device

PCIe SSD - P4510 (1TB)



BOSTON UNIVERSITY

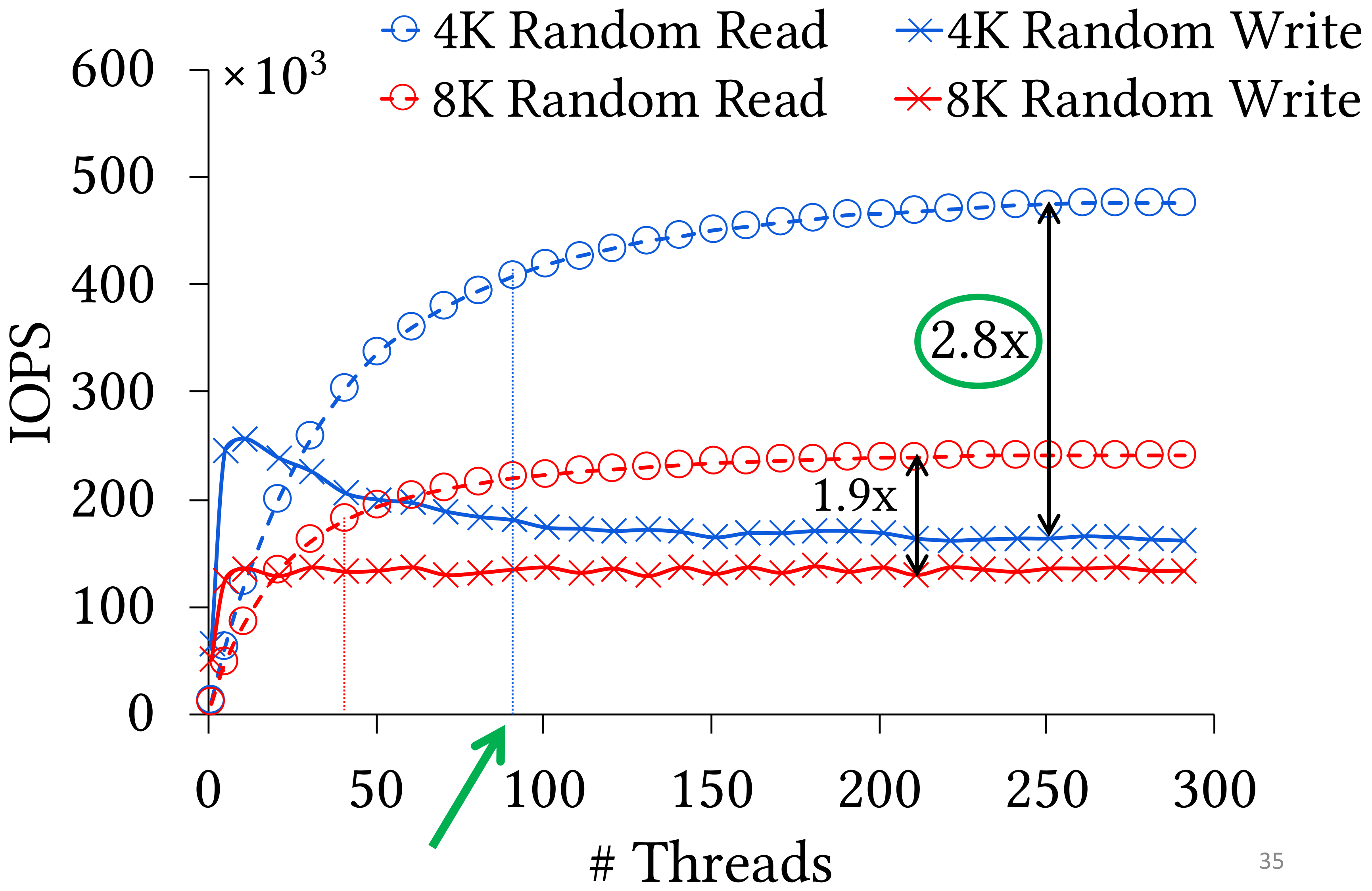
Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)

For 4K random read,

Asymmetry: 2.8

Concurrency: 80



BOSTON UNIVERSITY

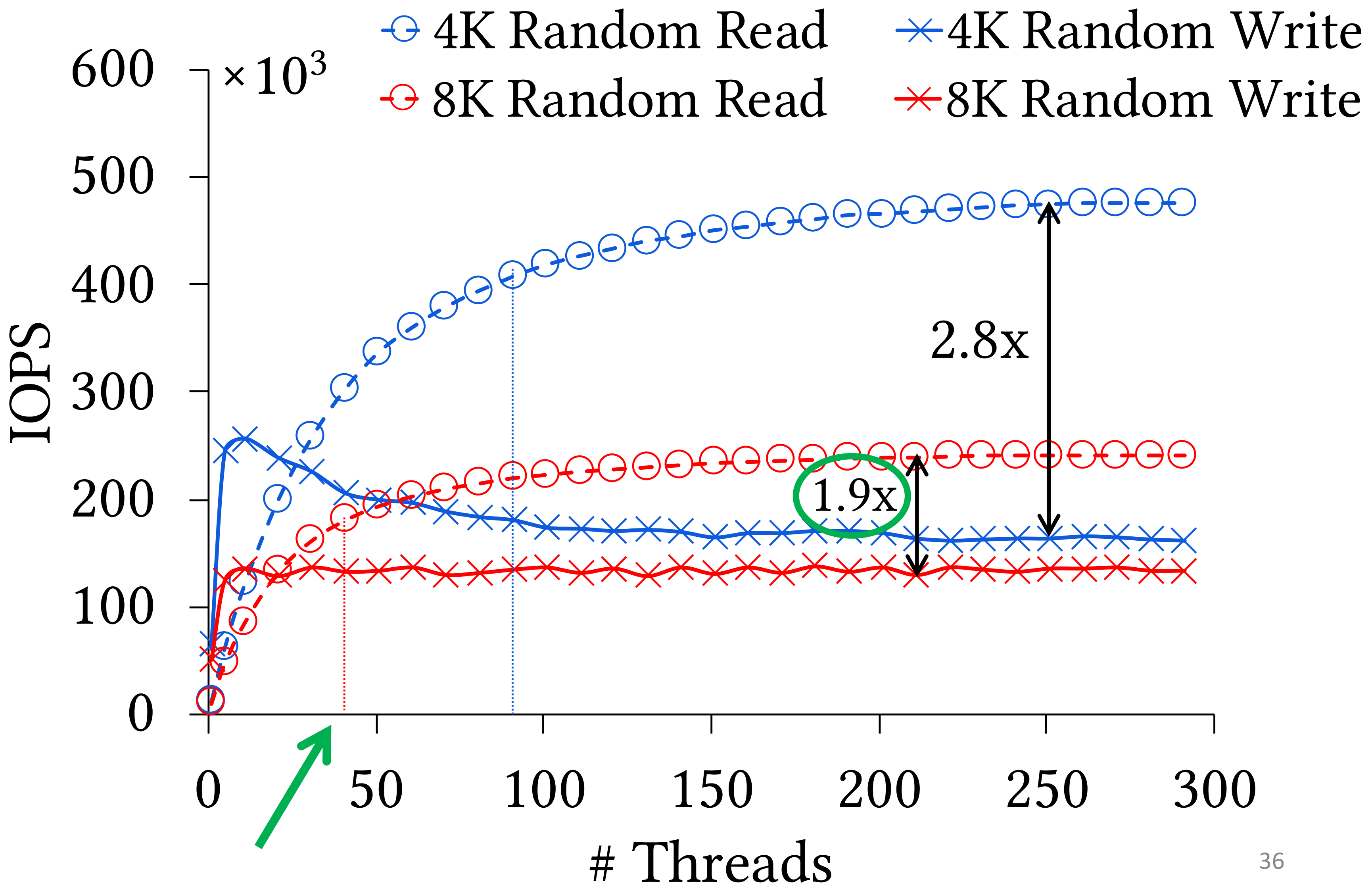
Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)

For 8K random read,

Asymmetry: 1.9

Concurrency: 40



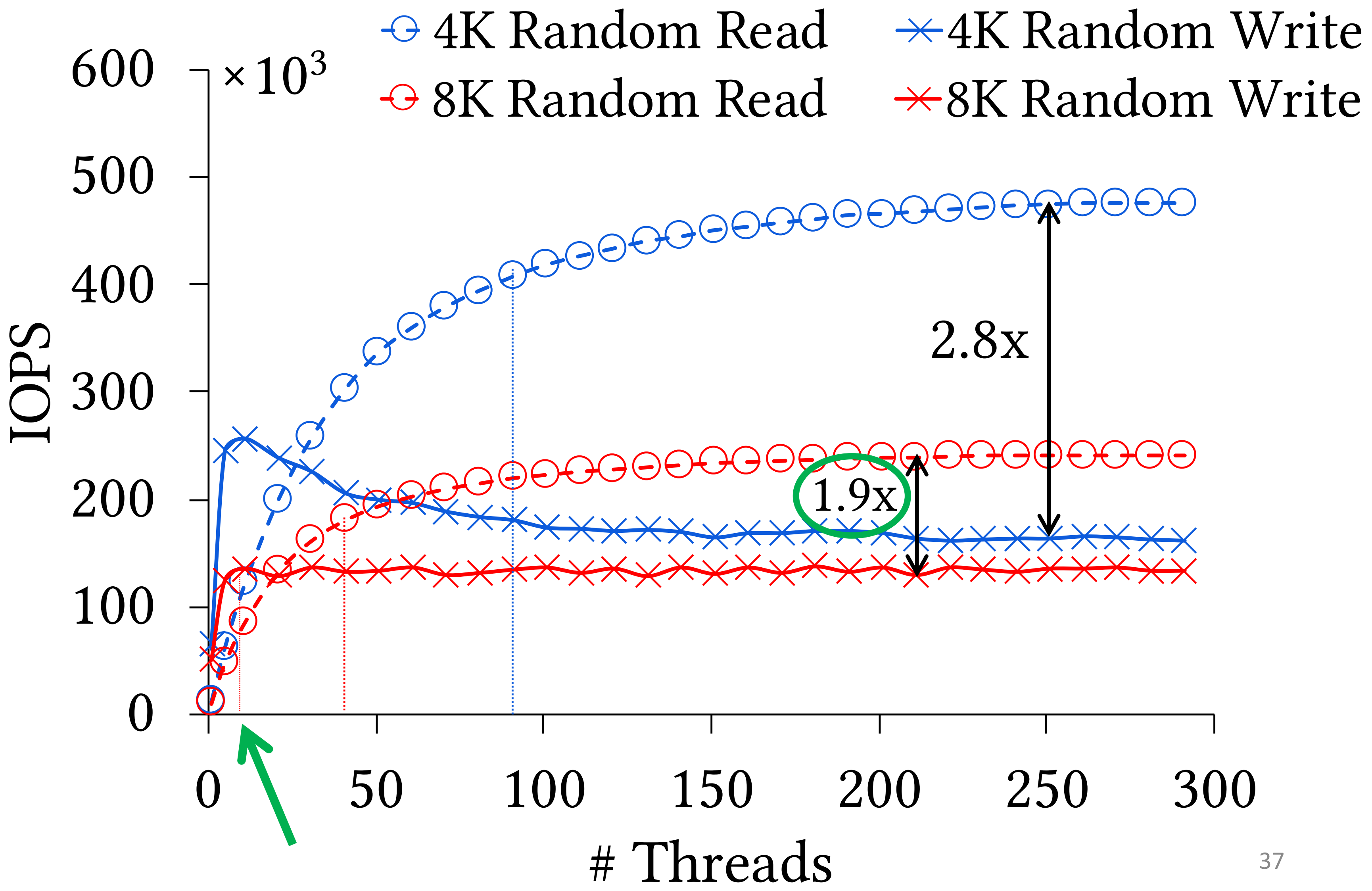
Measuring Asymmetry/Concurrency (With FS)

Device
PCIe SSD - P4510 (1TB)

For 8K random write,

Asymmetry: 1.9

Concurrency: 7

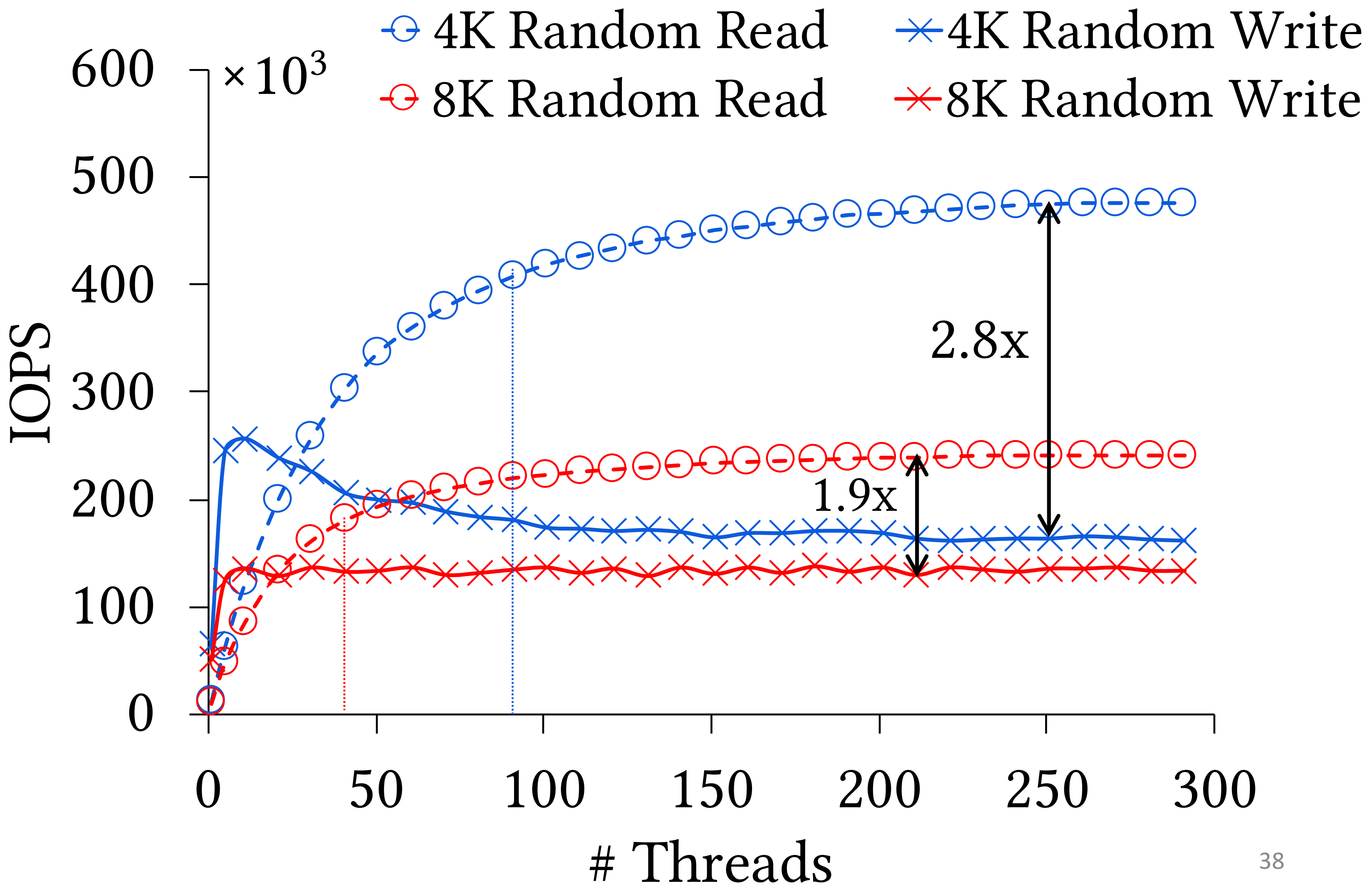


Measuring Asymmetry/Concurrency (With FS)

Device

PCIe SSD - P4510 (1TB)

Asymmetry and concurrency depends on request type and access granularity.



BOSTON UNIVERSITY

Empirical Asymmetry and Concurrency

	4KB			8KB		
Devices	α	k_r	k_w	α	k_r	k_w
Optane SSD	1.1	6	5	1.0	4	4
PCIe SSD (with FS)	2.8	80	8	2.0	40	7
PCIe SSD (w/o FS)	3.0	16	6	3.0	15	4
SATA SSD	1.5	25	9	1.3	21	5
Virtual SSD	2.0	>11	>19	1.9	>6	>10

BOSTON UNIVERSITY

Which module of a DBMS interacts with storage the most?

Bufferpool Manager

Buffer Pool Page Eviction Algorithm

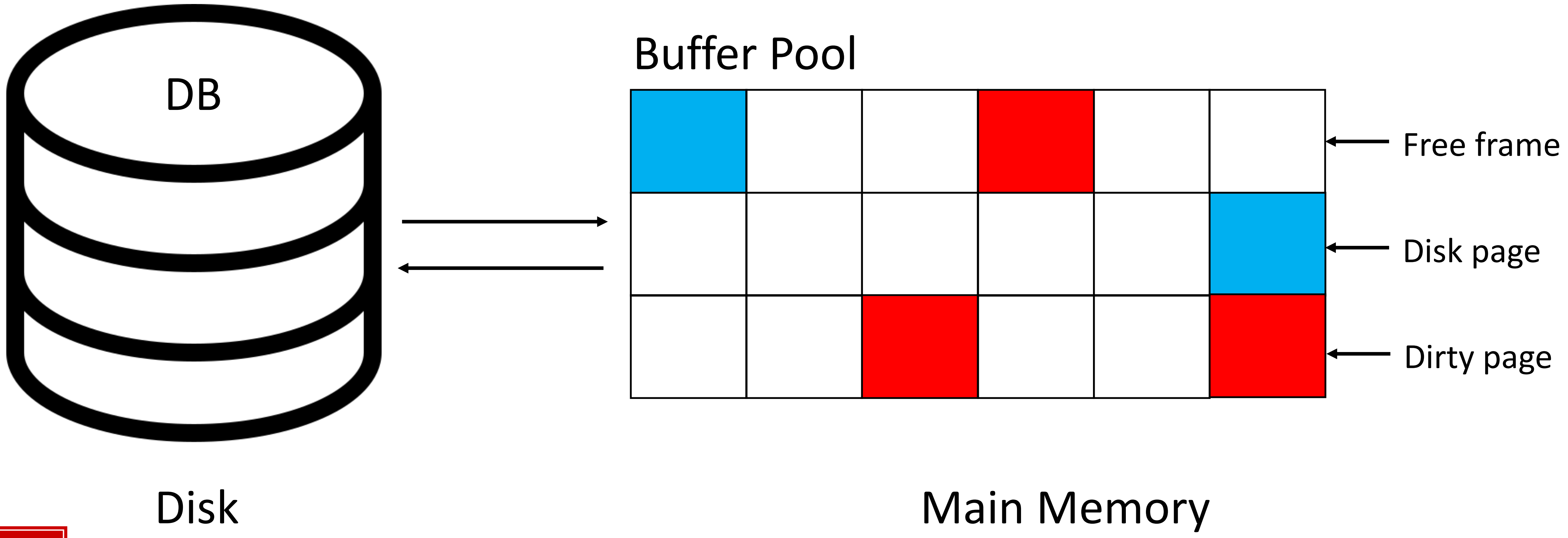
Classical

```
Request(page);
If (page in BP) -> return page
Else
    // Miss! Bring the page from Disk
    If BP not full -> Read requested page from Disk
    Else
        - Select a page for eviction based on replacement policy
        - If the candidate page is dirty, write to disk
        - Drop the candidate page from BP
        - Read requested page
```

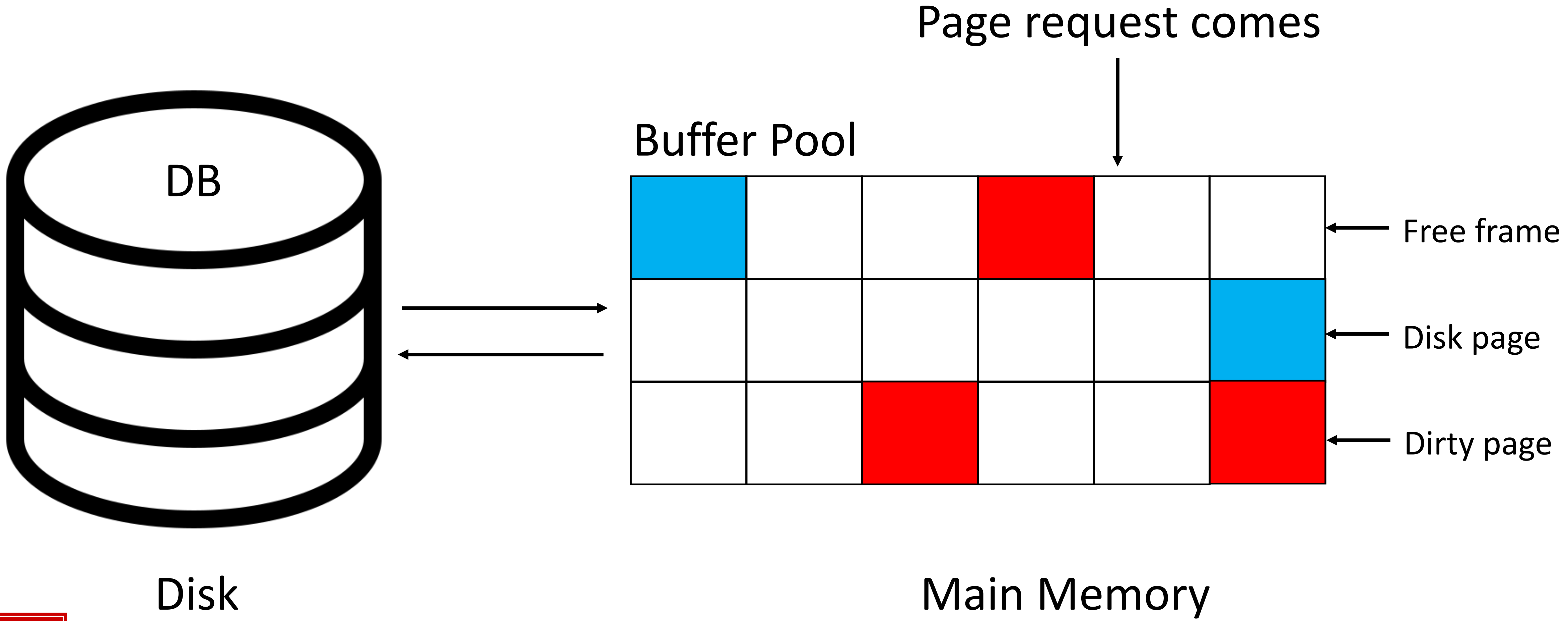
[if the request is a **write**, an **in-memory update** takes place that set the **dirty bit** as well]

BOSTON UNIVERSITY

Buffer Pool Manager

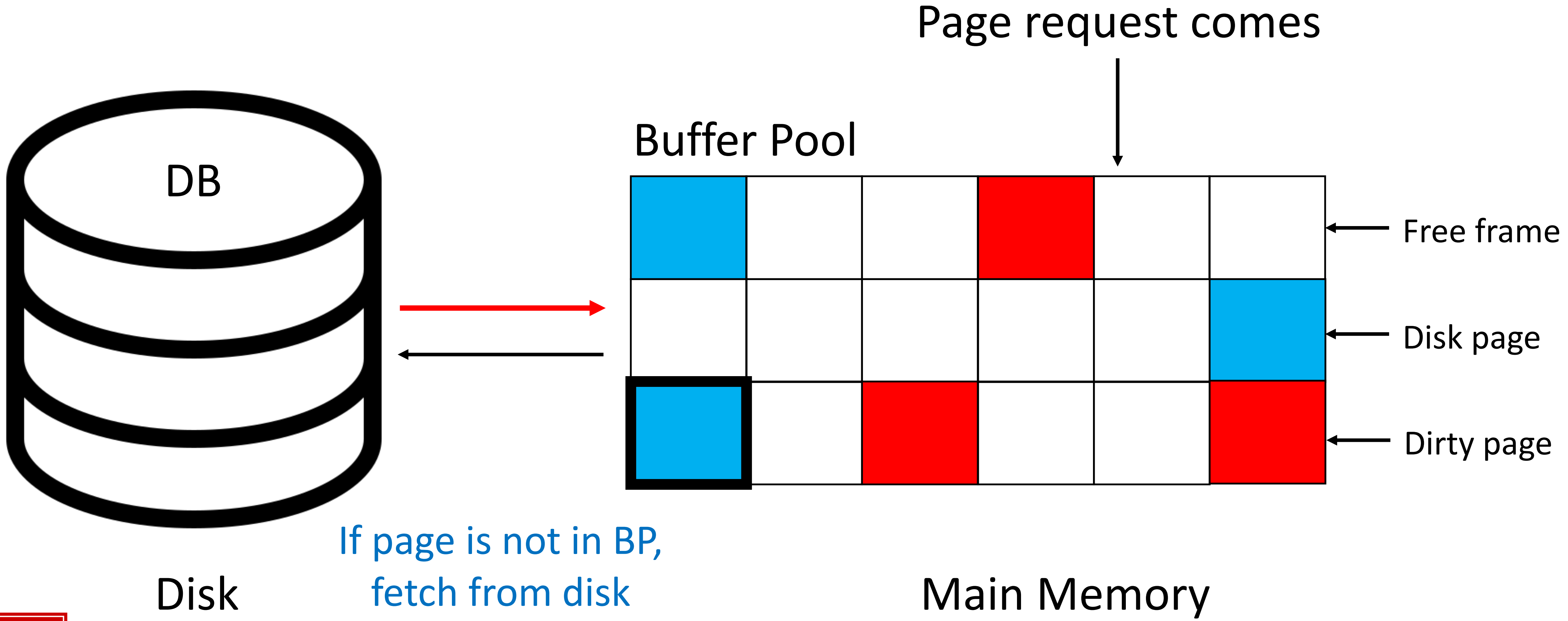


Traditional Buffer Pool Manager

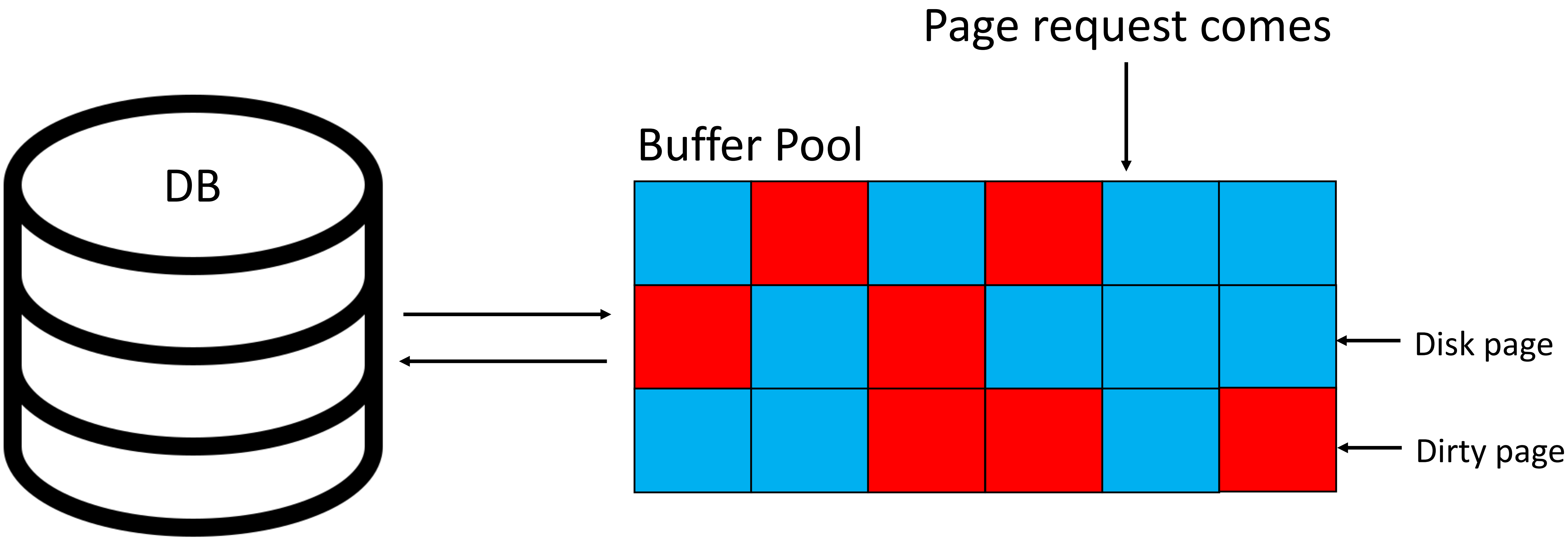


BOSTON UNIVERSITY

Traditional Buffer Pool Manager

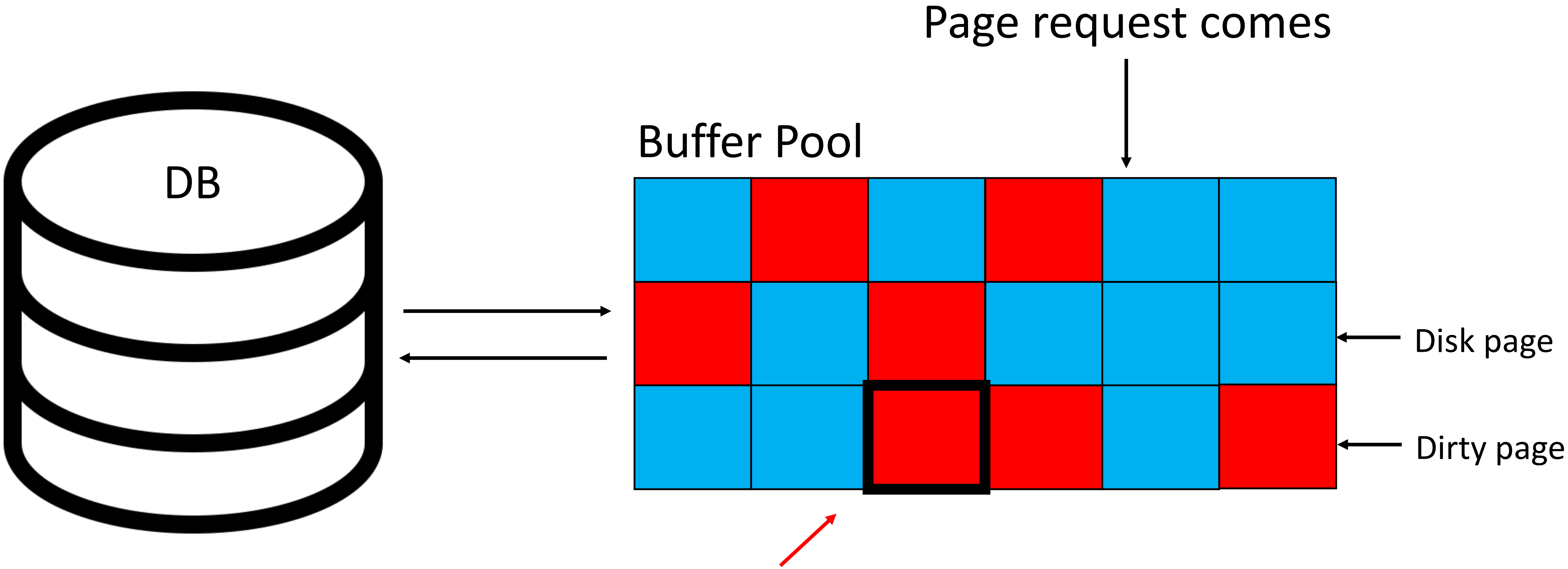


Traditional Buffer Pool Manager



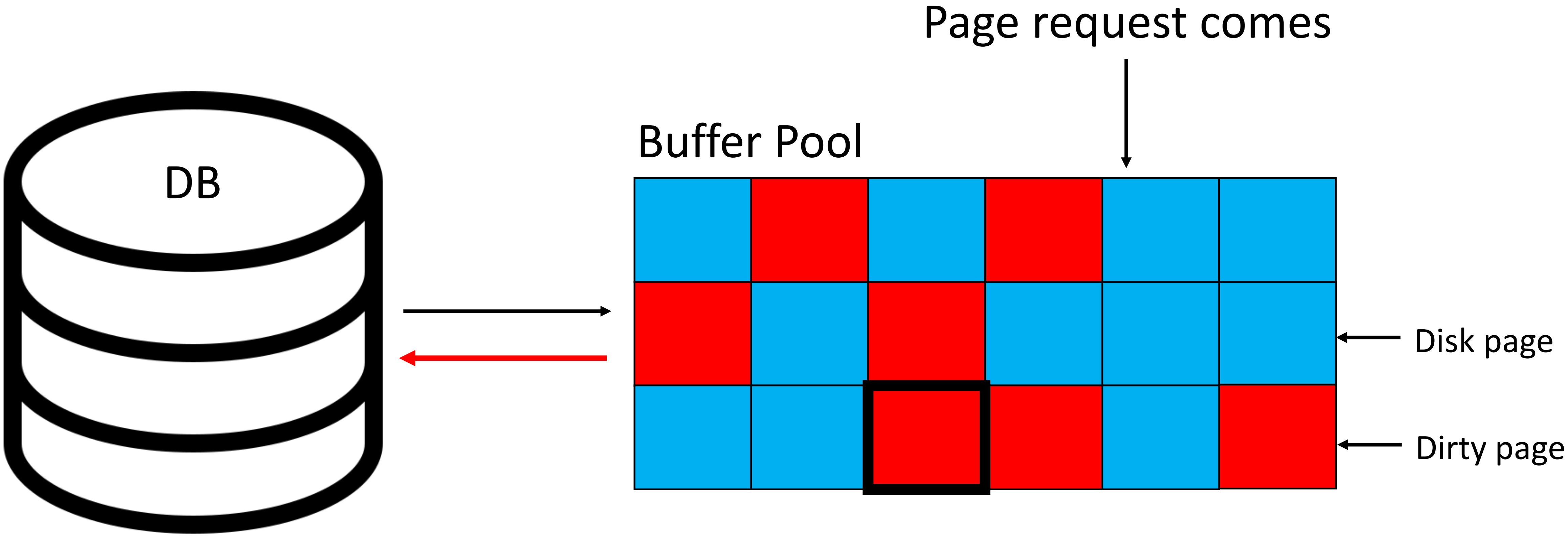
If BP is full, one page is selected for eviction based on **page replacement policy**

Traditional Buffer Pool Manager



If BP is full, one page is selected for eviction based on **page replacement policy**

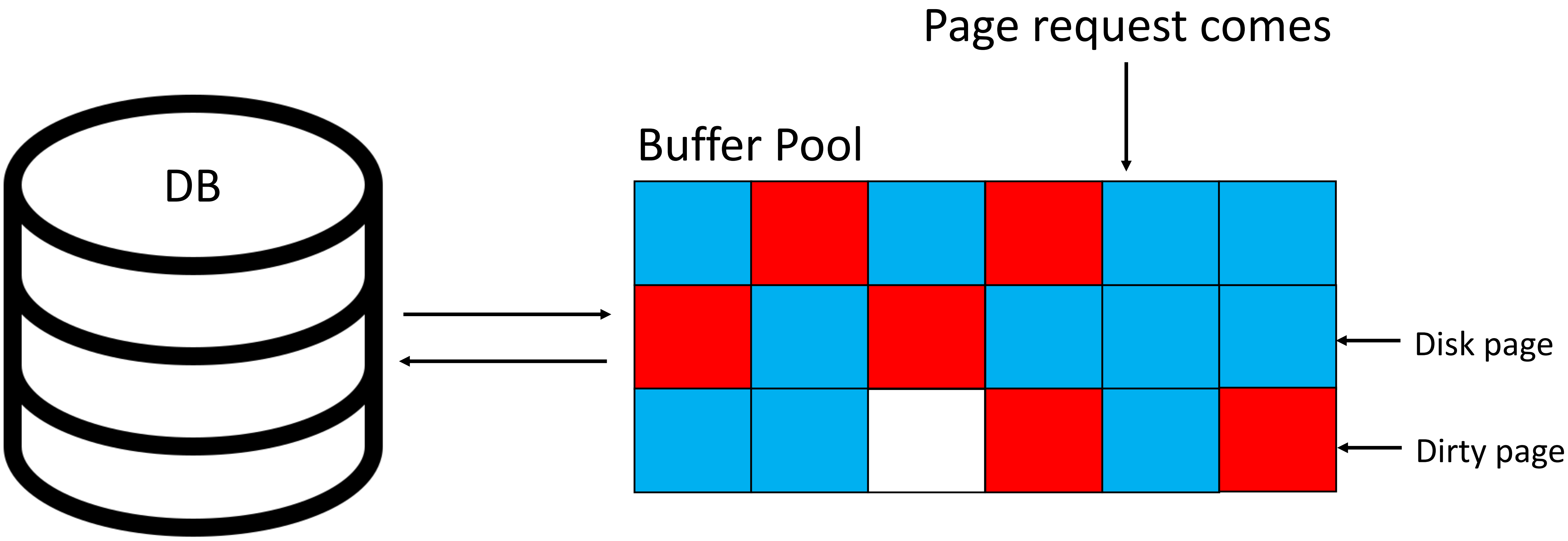
Traditional Buffer Pool Manager



If the page is dirty, it is **written back** to disk

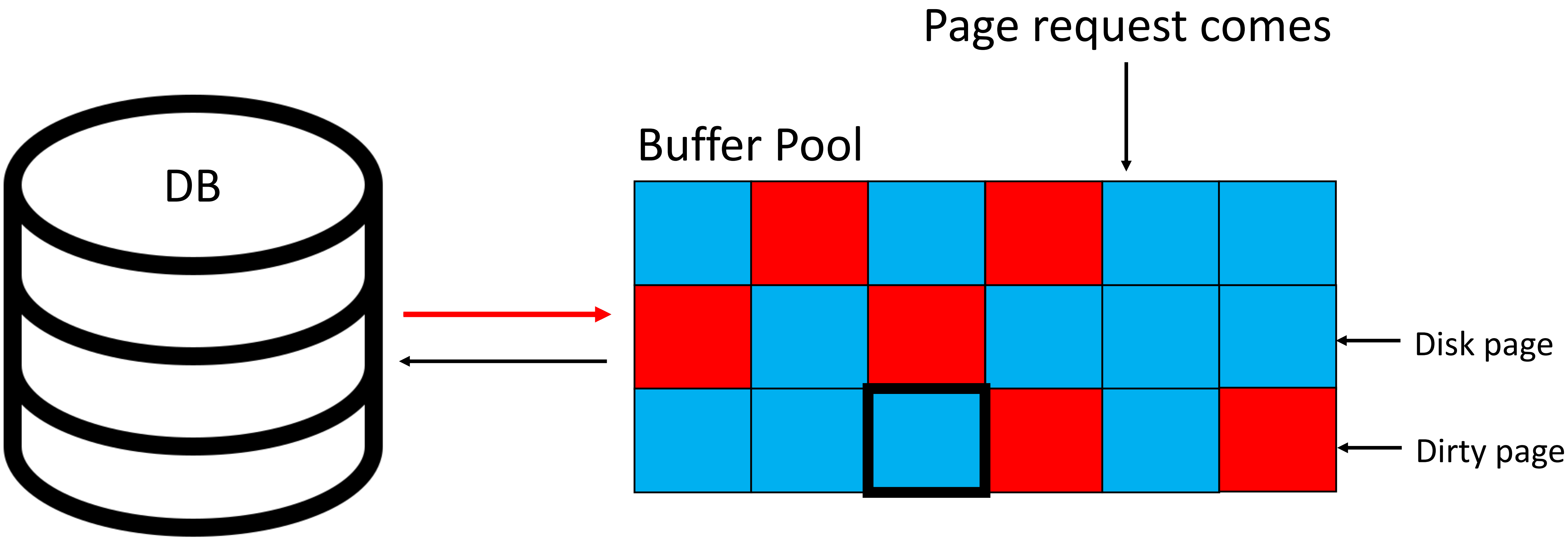
BOSTON UNIVERSITY

Traditional Buffer Pool Manager



If the page is dirty, it is **written back** to disk and **evicted**

Traditional Buffer Pool Manager



The requested page is fetched in its place

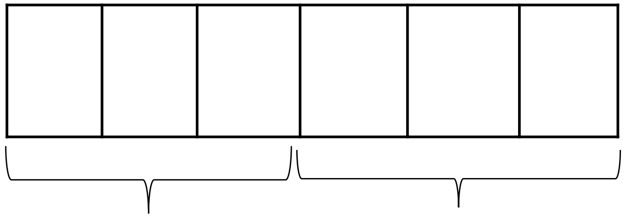
(exchanging one write for a read)

Popular Page Replacement Algorithms

LRU	(Most Popular)
LFU, FIFO	(Simple)
Clock Sweep	(Commercial)
CFLRU	Flash-Friendly
LRU-WSR	Flash-Friendly

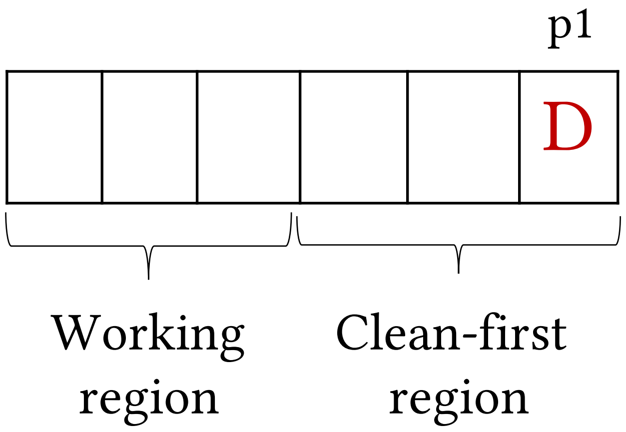
BOSTON UNIVERSITY

CFLRU

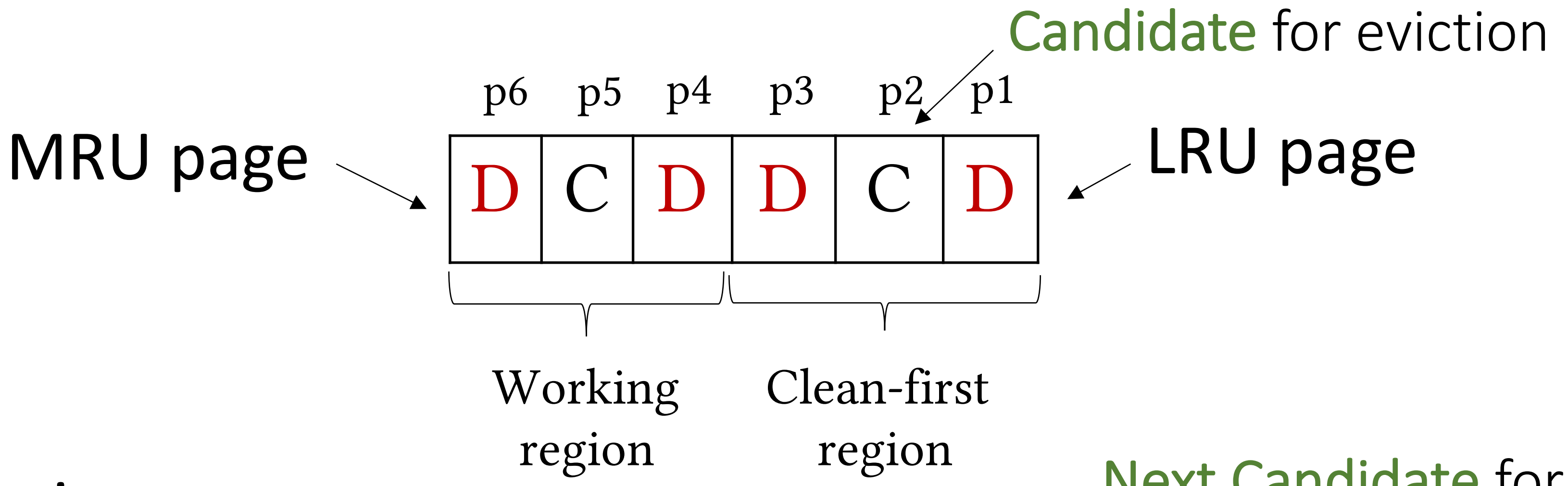


BOSTON UNIVERSITY

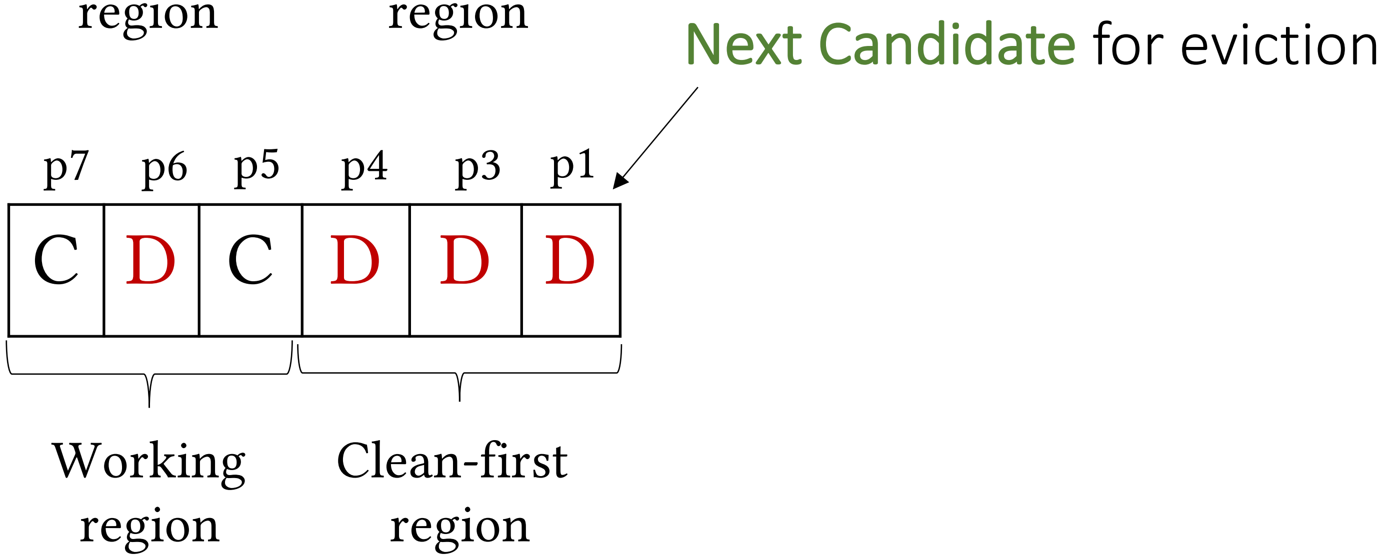
CFLRU



CFLRU



After Eviction:



LRU-WSR

	p6	p5	p4	p3	p2	p1
	D	C	D	D	C	D
Cold flag	1		0	0		0

← Cold flag NOT set! This is be moved to front setting the cold flag

	p1	p6	p5	p4	p3	p2
	D	D	C	D	D	C
Cold flag	1	1		0	0	

← Candidate for eviction

After Eviction:

	p7	p1	p6	p5	p4	p3
	C	D	D	C	D	D
Cold flag		1	1		0	0

BOSTON UNIVERSITY

LRU-WSR

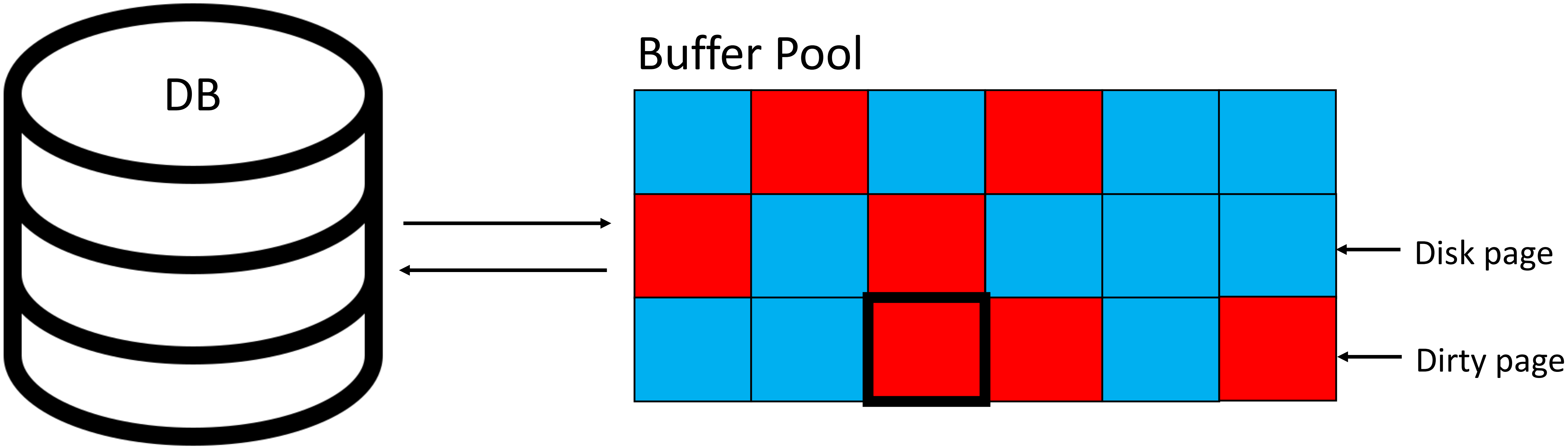
	p6	p5	p4	p3	p2	p1
	D	C	D	D	C	D
Cold flag	1		0	0		1

Cold flag set!
Candidate for eviction

After Eviction:

	p7	p6	p5	p4	p3	p2
	C	D	C	D	D	C
Cold flag		1		0	0	

The Challenge

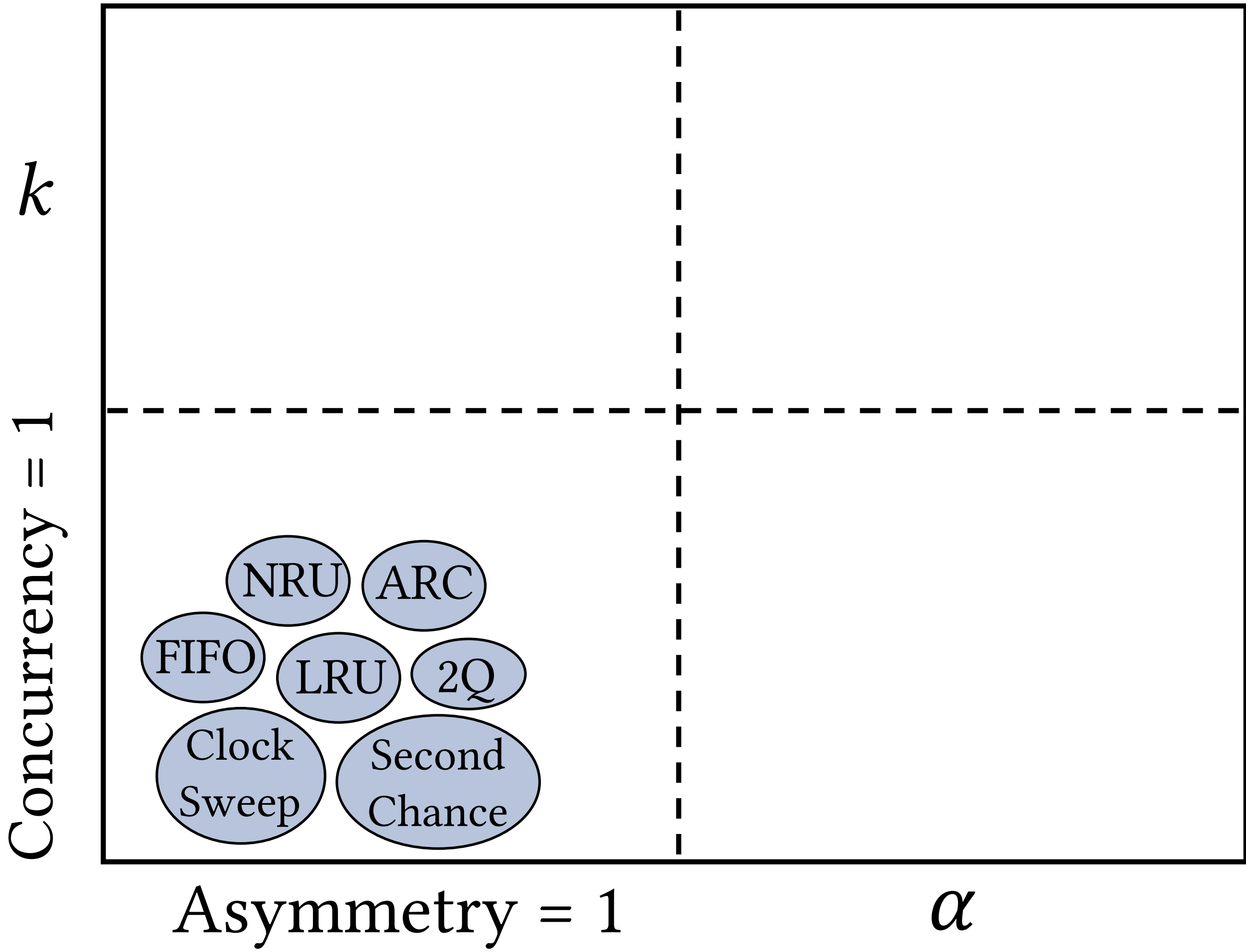


All these policies exchange one read for one write!

Is this Fair?

The Challenge

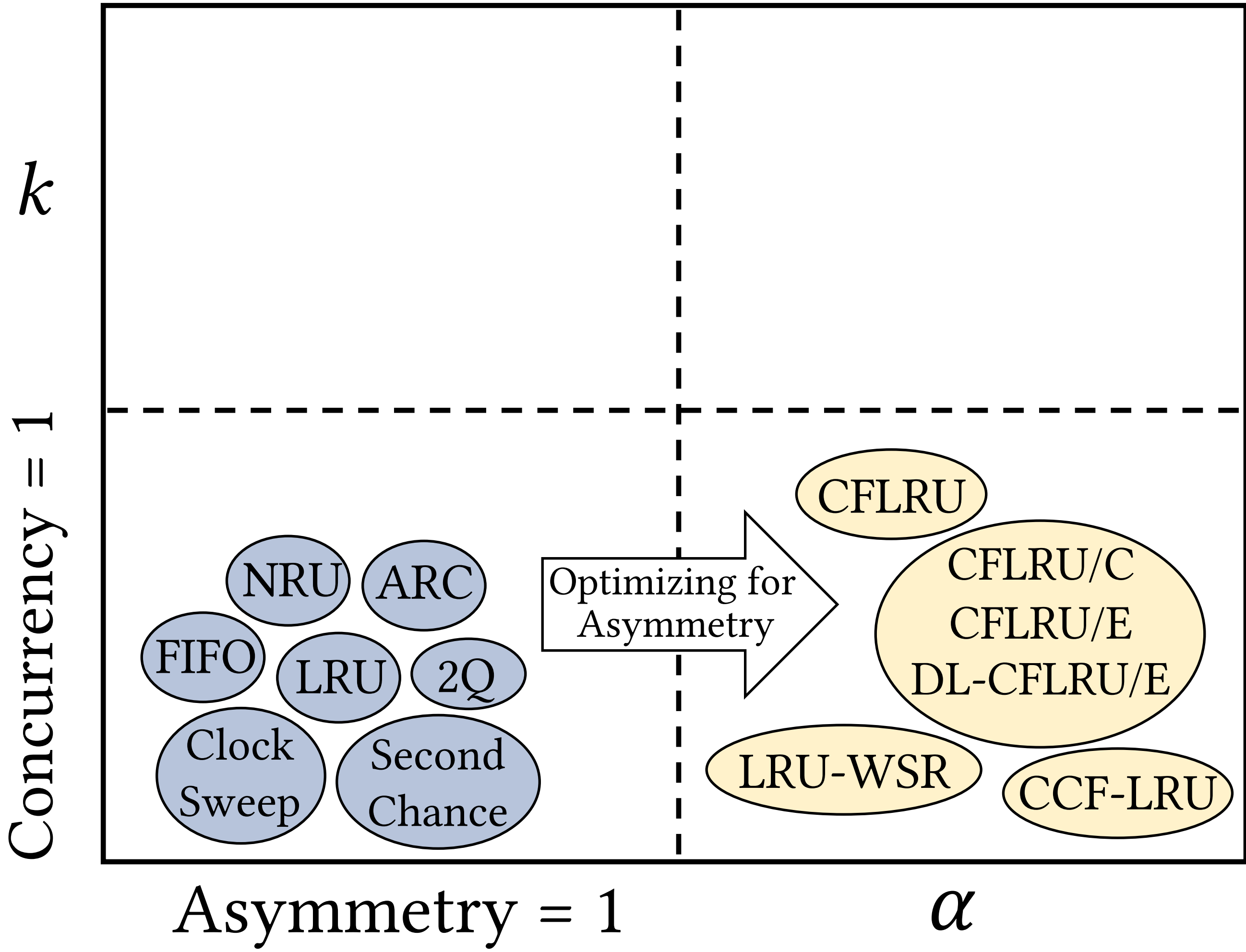
- (Challenge 1) With write asymmetry, it is **NOT** fair to *exchange one write for one read*.



BOSTON UNIVERSITY

The Challenge

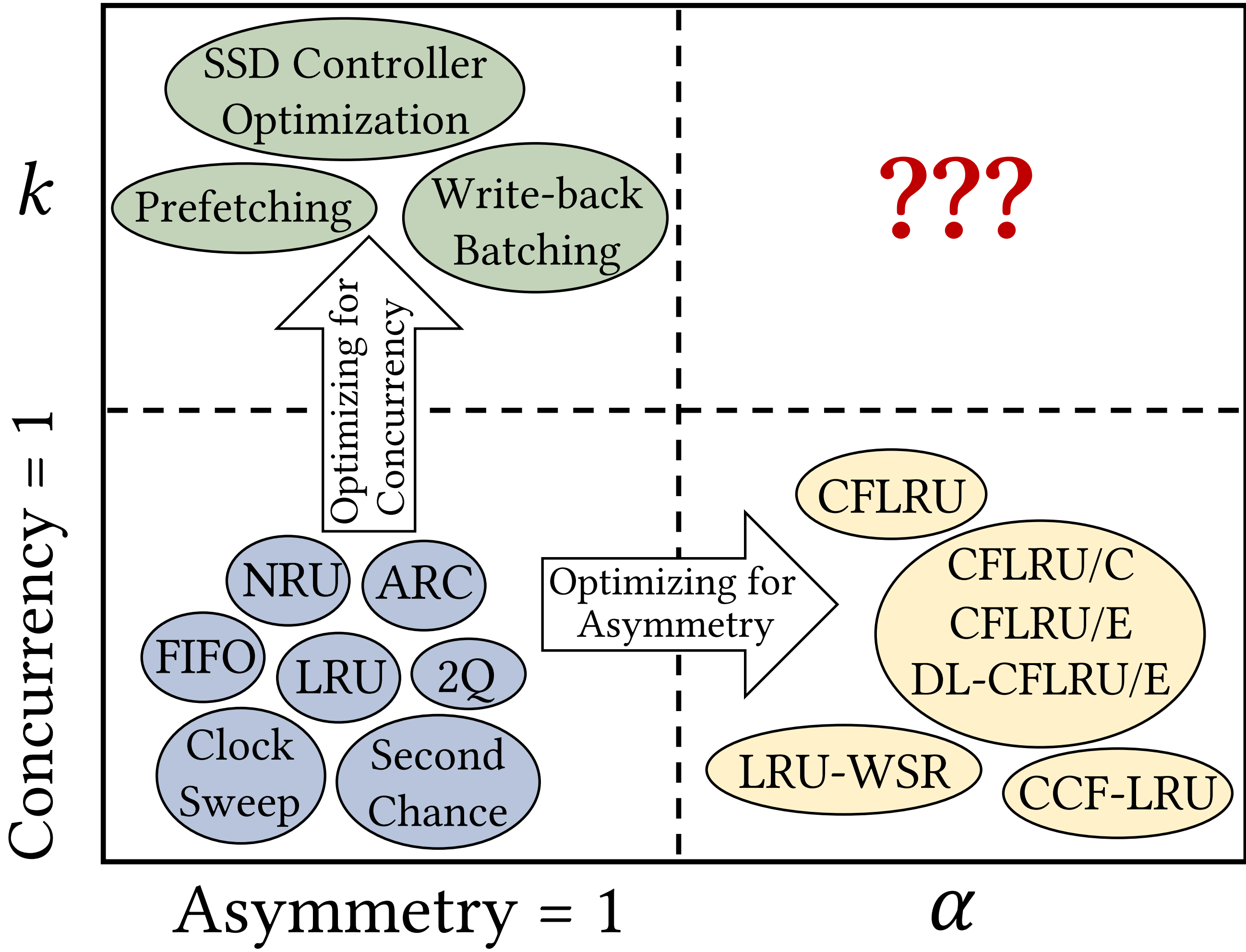
- (Challenge 1) With write asymmetry, it is **NOT** fair to *exchange one write for one read*.



BOSTON UNIVERSITY

The Challenge

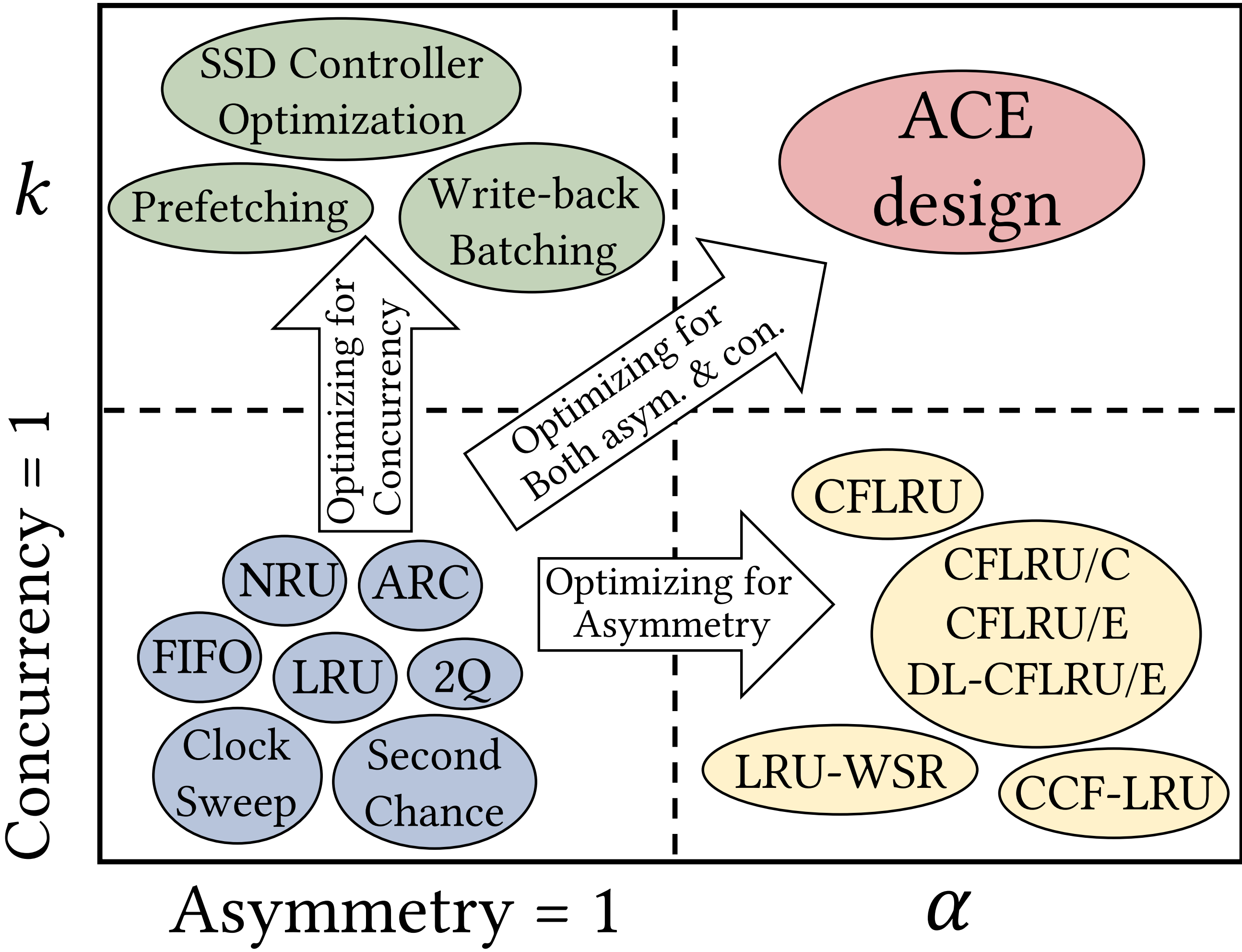
- (Challenge 1) With write asymmetry, it is **NOT** fair to *exchange one write for one read*.
- (Challenge 2) Bufferpool Managers do **NOT** expressly utilize the *device concurrency*.



BOSTON UNIVERSITY

The Challenge

- (Challenge 1) With write asymmetry, it is **NOT** fair to *exchange one write for one read*.
- (Challenge 2) Bufferpool Managers do **NOT** expressly utilize the *device concurrency*.



Bufferpool Design Space

Classical Bufferpool Design Space

Bufferpool Manager

Eviction Policy

Which page to evict?

- LRU
- NRU
- Clock
- Second Chance
- FIFO
- 2Q
- ARC

- CFLRU
- LRU-WSR
- CCF-LRU
- CFLRU/C
- CFLRU/E
- DL-CFLRU/E

Flash-friendly policies

Read-ahead Policy (Optional)

When to prefetch?

- Prefetch on miss

Which pages?

- Sequential/
- History-based

How many pages?

- 1 or x pages

How to prefetch?

- Sequentially

Bufferpool Design Space

Bufferpool Manager

Red indicates new design elements

Eviction Policy

How many pages to evict?

Which pages to evict?

- LRU
- NRU
- Clock
- Second Chance
- FIFO
- 2Q
- ARC

Flash-friendly policies:

- CFLRU
- LRU-WSR
- CCF-LRU
- CFLRU/C
- CFLRU/E
- DL-CFLRU/E

Write-back Policy

How many pages to flush?

- 1 page
- *n* pages (exploit k_w)

Which pages to flush?

- dirty pages following replacement algo.

How to write?

- one-at-a-time vs. concurrently

Read-ahead Policy (Optional)

When to prefetch?

- Prefetch on miss

Which pages?

- Sequential
- History-based

How many pages?

- 1 or *x* pages

How to prefetch?

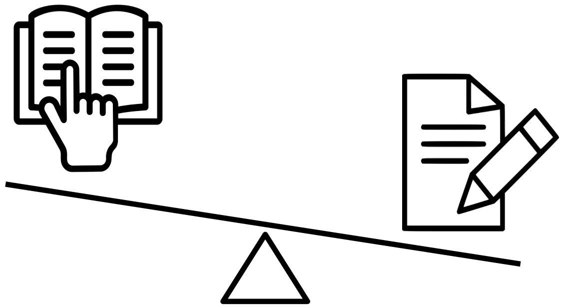
- Seq./Concurrently

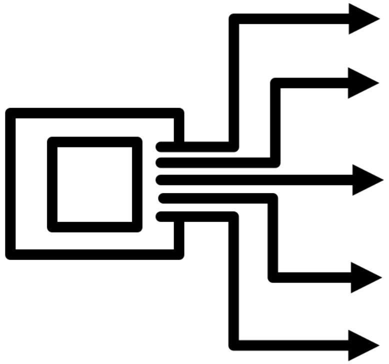
Asymmetry/Concurrency-Aware (ACE) Bufferpool Manager

BOSTON UNIVERSITY

ACE Bufferpool Manager

Use device's properties





BOSTON UNIVERSITY

ACE Bufferpool Manager

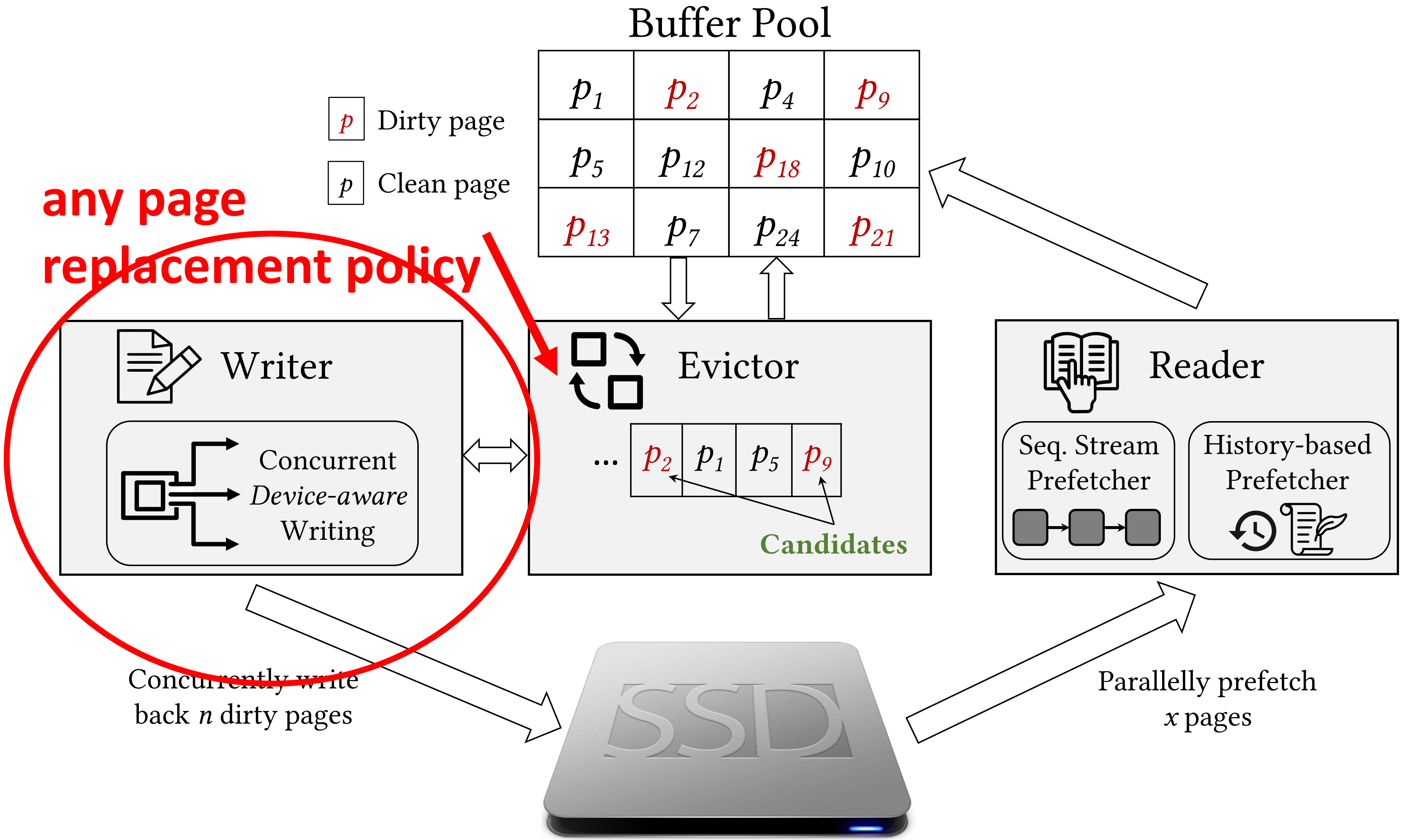
Flush multiple **dirty** pages **concurrently**

Evict 1 page (to not disrupt locality)
or
Evict multiple pages (if we trust prefetching)

Goal: 1 read vs. α concurrent write backs (more if concurrency permits)

ACE Bufferpool Manager

any page replacement policy



with PF

w/o PF

Buffer Pool Page Eviction Algorithm

```
Request(page pR);
If (page in BP) -> return page
Else
    // Miss! Bring the page from Disk
    If BP not full -> Read requested page from Disk
    Else
        - Select a page pE for eviction (?) based on replacement policy
        - …
```

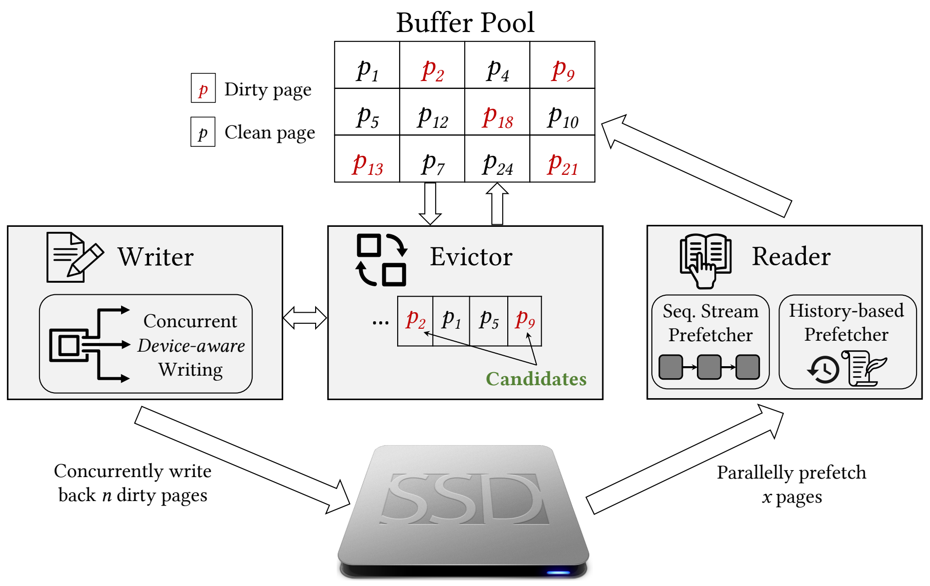
BOSTON UNIVERSITY

Module: Writer

Concurrent write-back

- If $\mathbf{p_E}$ is dirty, then write *n* dirty pages concurrently

  where, *n* = device's write concurrency (k_w)
- If $\mathbf{p_E}$ is clean, skip to Evictor



The *n* pages are selected following the order of the underlying page replacement algorithm

Module: Evictor

Piggy-backs on the underlying replacement algorithm

If **prefetching is not enabled**

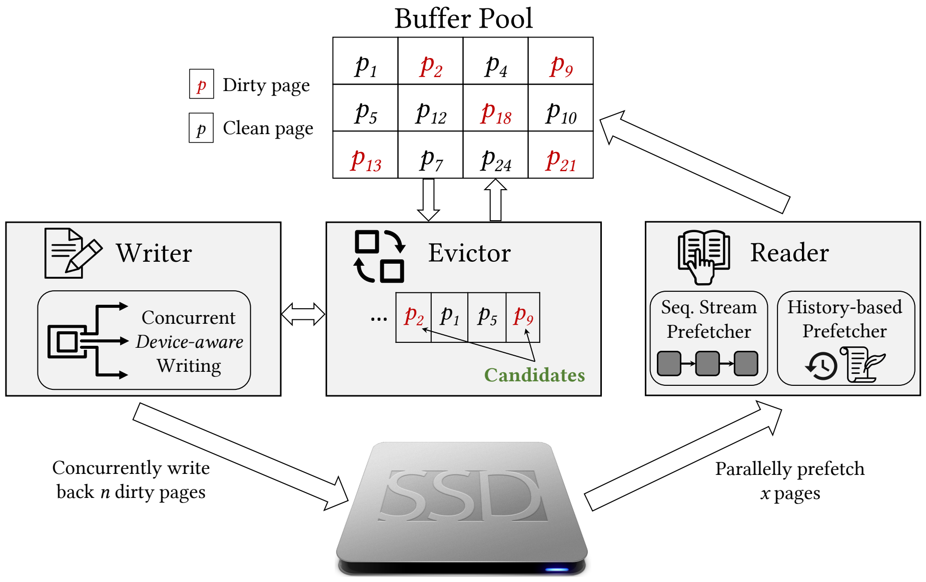
✓ Evict 1 page (following the replacement algorithm)

If **prefetching is enabled** & $\mathbf{p_E}$ is clean

✓ Evict 1 page

If **prefetching is enabled** & $\mathbf{p_E}$ is dirty

✓ Evict n pages



BOSTON UNIVERSITY

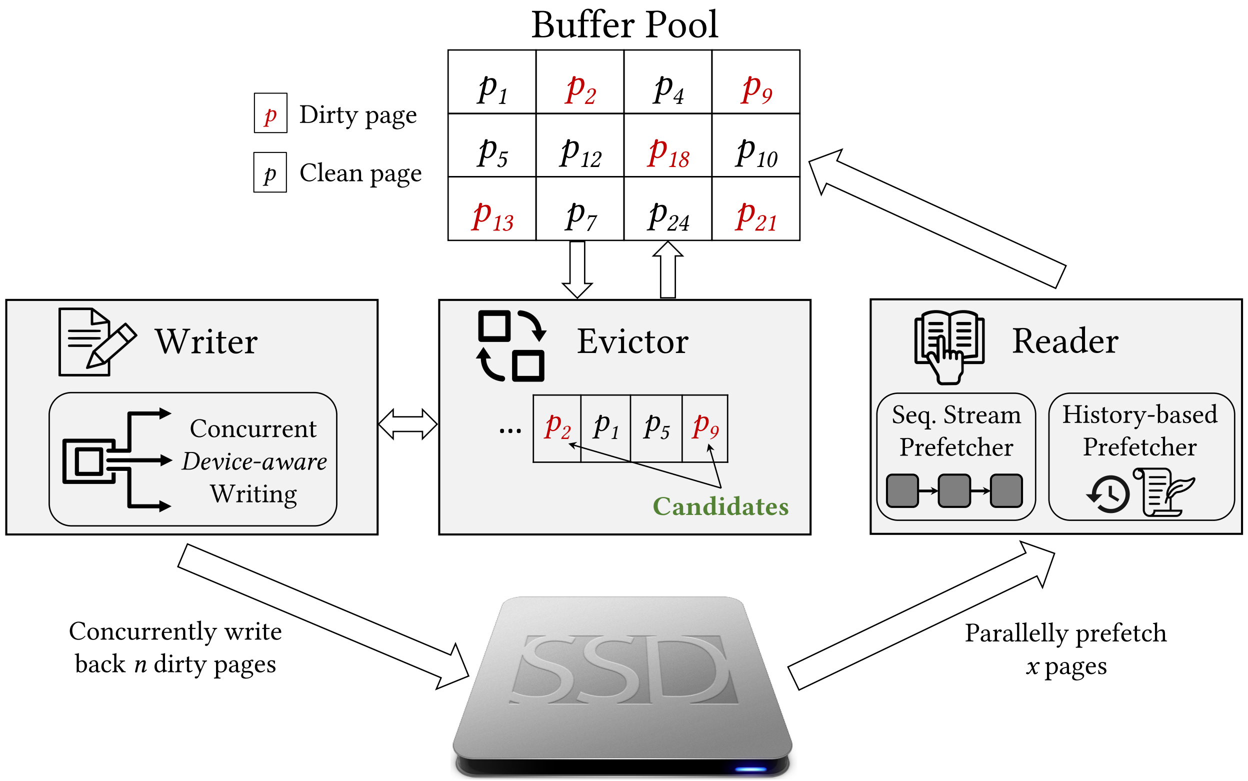
Module: Reader

Fetch the **requested page $\mathbf{p_E}$**

If prefetching is enabled (n pages were evicted)

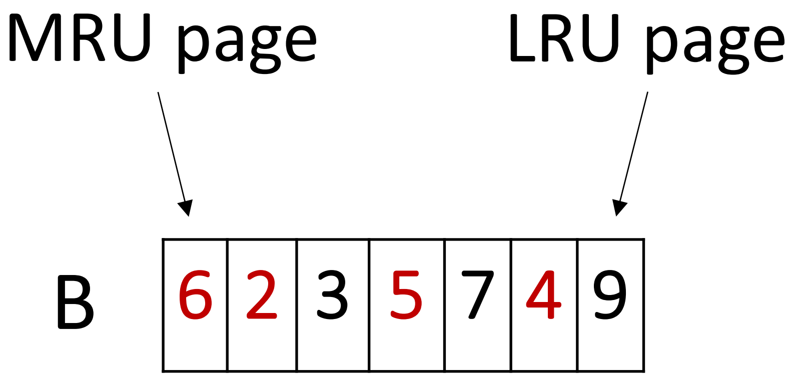
✓ Prefetch **concurrently** *n-1* pages

- Sequential prefetcher
- History-based prefetcher



BOSTON UNIVERSITY

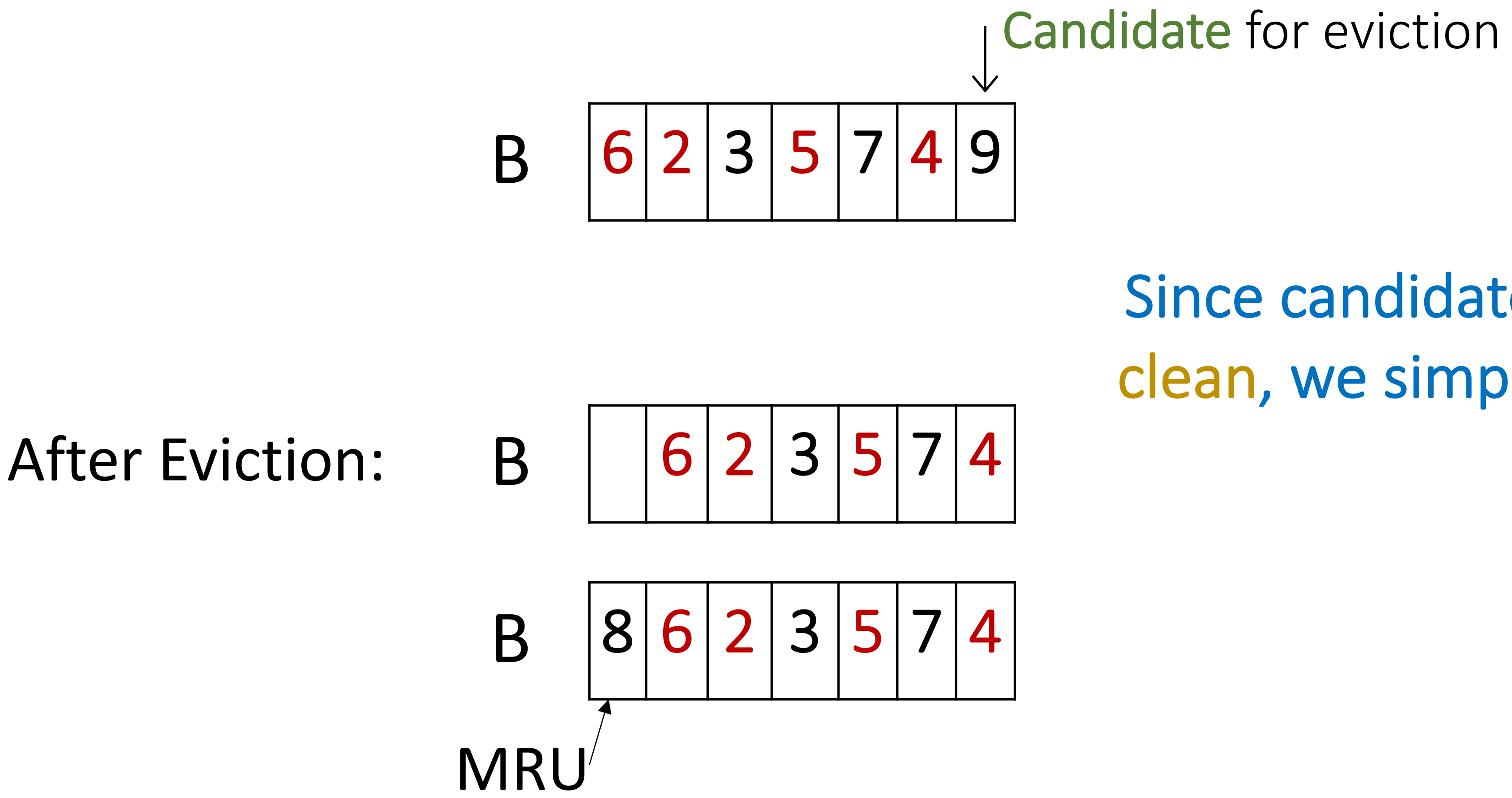
Let’s Take a Look at an Example



Let’s assume: $\alpha = 3$, LRU is the baseline replacement policy & red indicates dirty page

Next, a read request for page 8 arrives

Let's Take a Look at an Example



Since candidate page is clean, we simply evict it

Next, a write request for page 1 arrives

Let's Take a Look at an Example ($\alpha = 3$)

LRU

B	8	6	2	3	5	7	4 (Candidate)

Eviction:

evict candidate (4)

B		8	6	2	3	5	7
B	1	8	6	2	3	5	7

Let’s Take a Look at an Example (α = 3)

LRU

Candidate ↓ (4)

B	8	6	2	3	5	7	4

Eviction:
evict candidate (4)

B		8	6	2	3	5	7

B	1	8	6	2	3	5	7

ACE LRU(w/o PF)

Candidate ↓ (4)

B	8	6	2	3	5	7	4

Write-back: 2,5,4 concurrently written

Eviction: 4 is evicted

B		8	6	2	3	5	7

B	1	8	6	2	3	5	7

Let’s Take a Look at an Example ($\alpha = 3$)

LRU

Candidate ↓

B | 8 | 6 | 2 | 3 | 5 | 7 | 4 |

Eviction:
evict candidate (4)

B | | 8 | 6 | 2 | 3 | 5 | 7 |

B | 1 | 8 | 6 | 2 | 3 | 5 | 7 |

ACE LRU(w/o PF)

Candidate ↓

B | 8 | 6 | 2 | 3 | 5 | 7 | 4 |

Write-back: 2,5,4 concurrently written

Eviction: 4 is evicted

B | | 8 | 6 | 2 | 3 | 5 | 7 |

B | 1 | 8 | 6 | 2 | 3 | 5 | 7 |

ACE LRU(w/ PF)

Eviction Window ↓

B | 8 | 6 | 2 | 3 | 5 | 7 | 4 |

Write-back: 2,5,4 concurrently

Eviction: 5,7,4 is evicted

Prefetch: 9,0 (as LRU)

B | | 8 | 6 | 2 | 3 | 9 | 0 |

B | 1 | 8 | 6 | 2 | 3 | 9 | 0 |

Evaluation

Experimental Evaluation

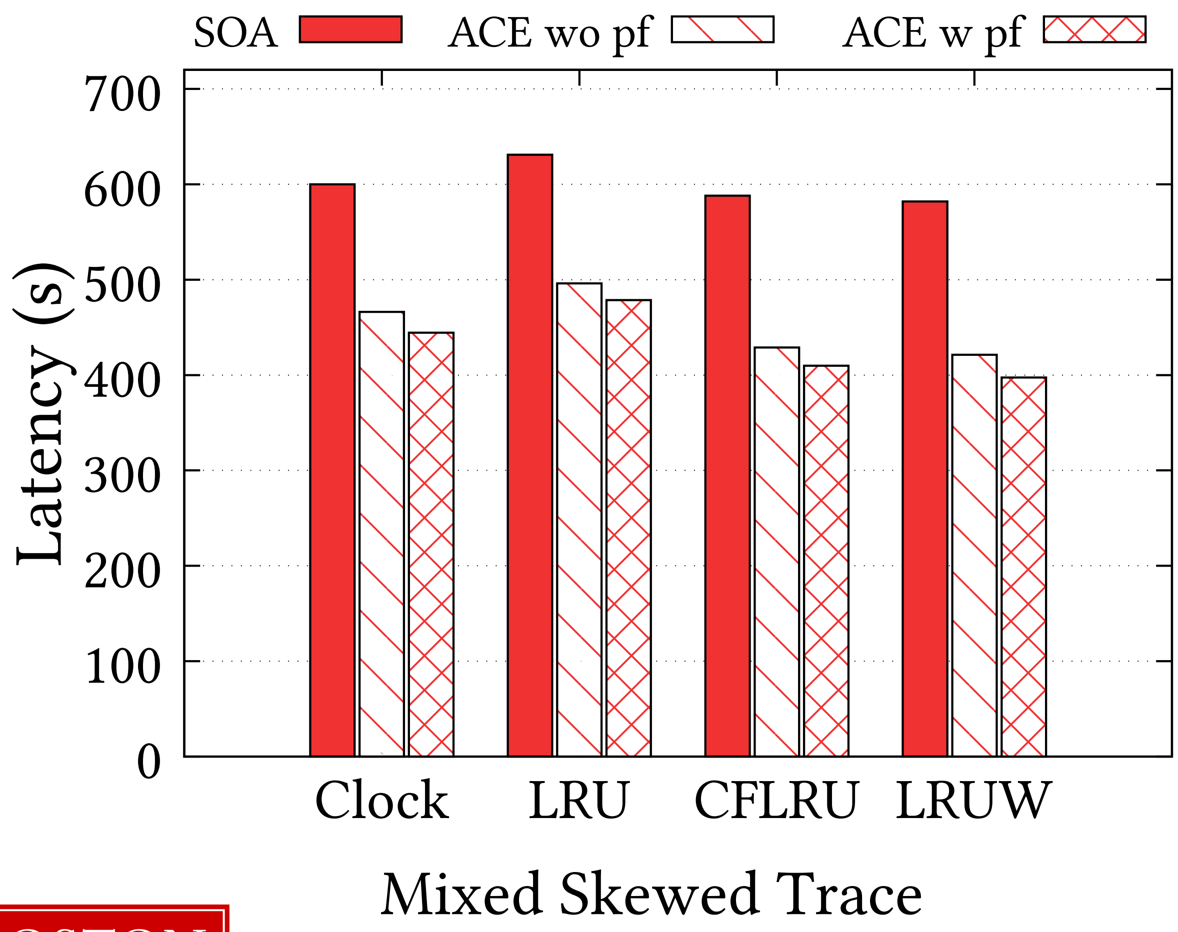
- Implementation in PostgreSQL
- Clock, LRU, CFLRU, LRU-WSR vs. their ACE counterparts
- Evaluation on 4 synthesized traces and TPC-C benchmark
- 3 storage devices: NVMe SSD, $\alpha = 3$, $k_w = 8$; Regular SSD, $\alpha = 1.5$, $k_w = 8$; Virtual SSD, $\alpha = 2$, $k_w > 19$

BOSTON UNIVERSITY

Experimental Evaluation

Device: NVMe SSD

$\alpha = 3$, $k_w = 8$



ACE improves runtime by >20%

(for a mixed skewed workload)

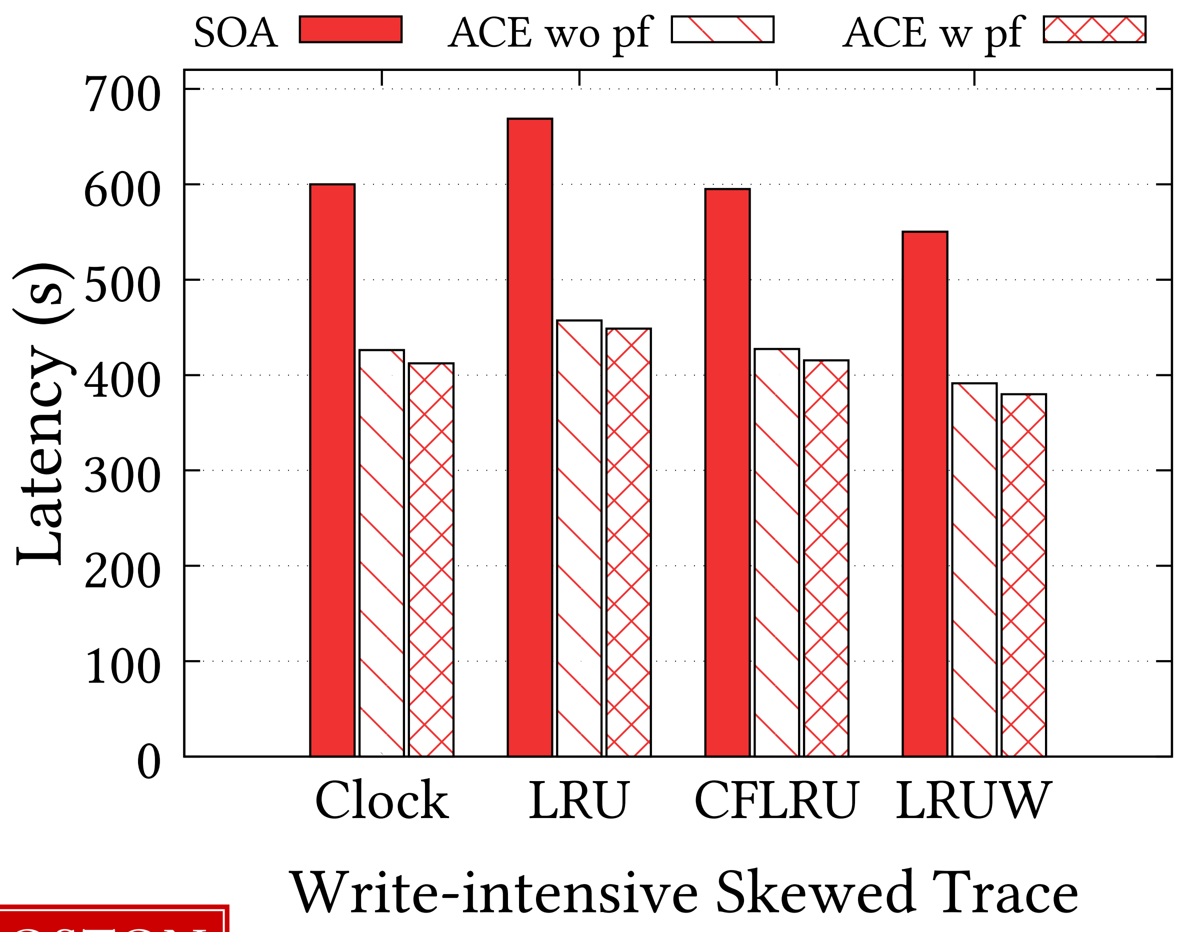
Negligible increase in buffer miss (<0.004%)

BOSTON UNIVERSITY

Experimental Evaluation

Device: NVMe SSD

$\alpha = 3$, $k_w = 8$



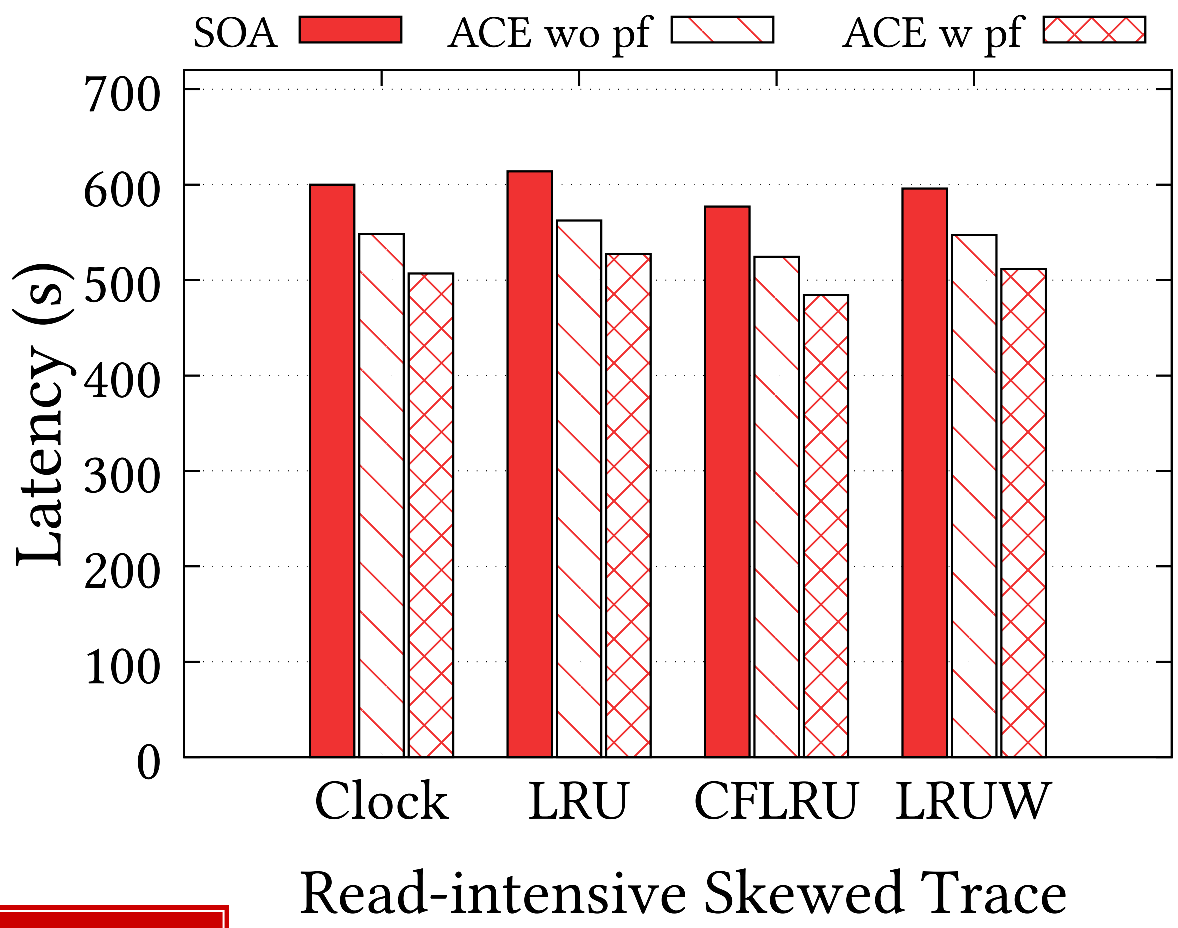
Higher gain (>30%) for write-intensive workloads because of smart batching

BOSTON UNIVERSITY

Experimental Evaluation

Device: NVMe SSD

$\alpha = 3, k_w = 8$



Read-intensive workloads also have substantial gain (5-15%)

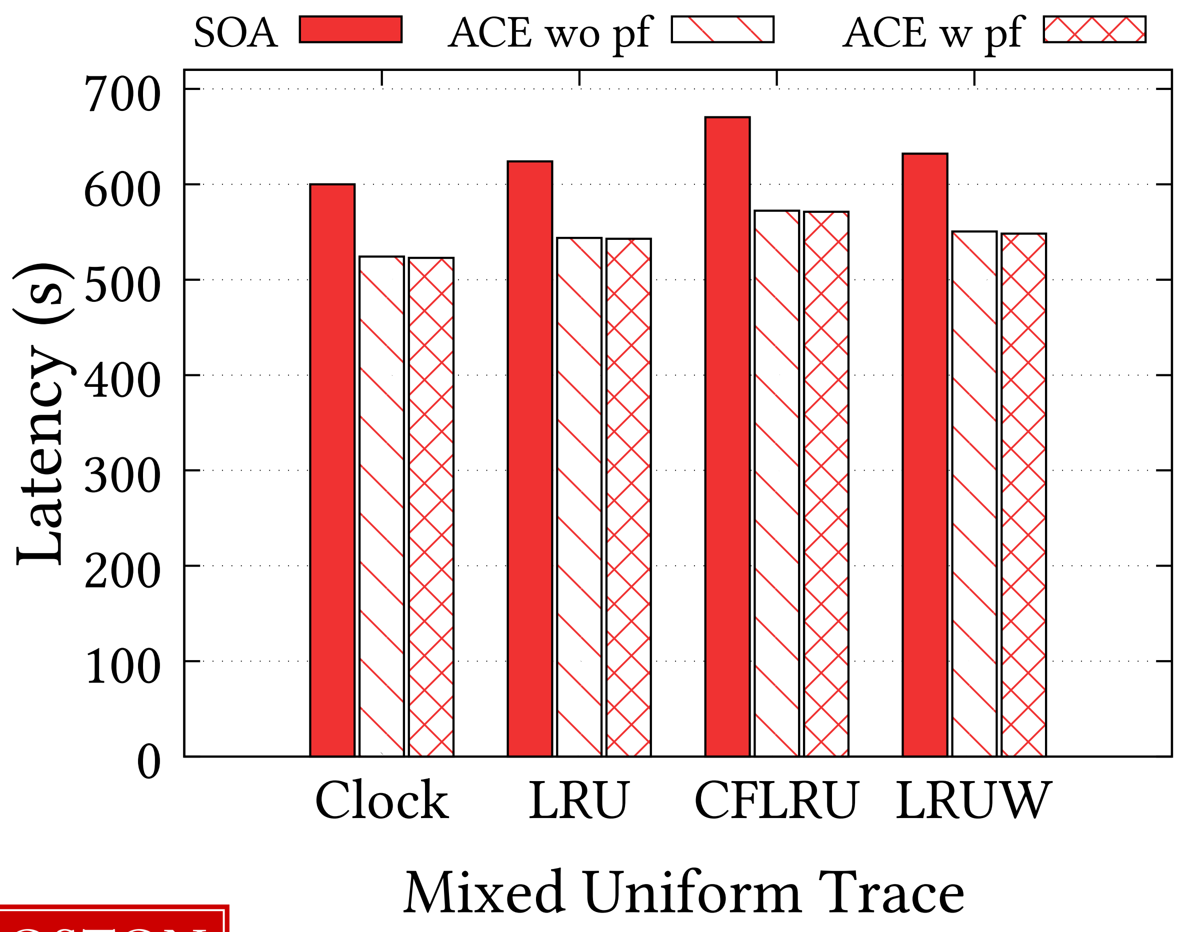
Benefits comes mostly through prefetching

BOSTON UNIVERSITY

Experimental Evaluation

Device: NVMe SSD

$\alpha = 3, k_w = 8$



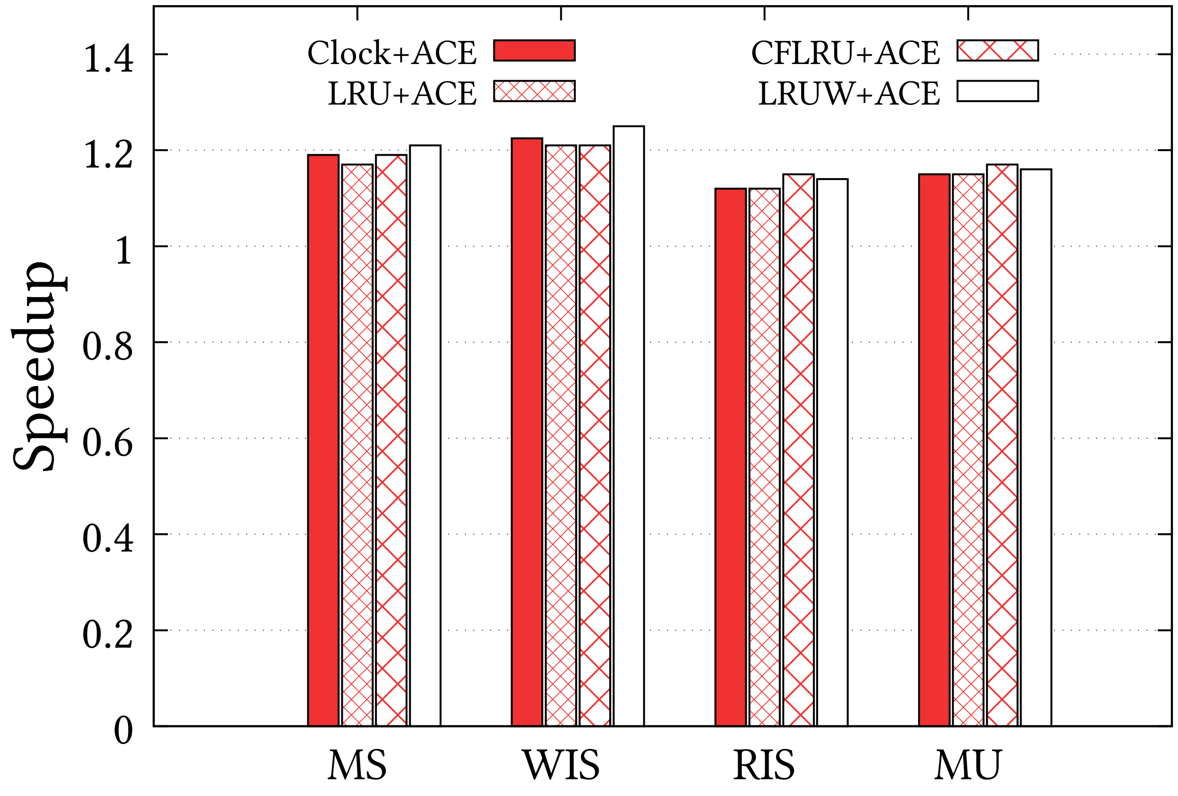
The benefits remain significant (~10%) for uniform workloads

BOSTON UNIVERSITY

Experimental Evaluation

Device: Regular SSD

$\alpha = 1.5$, $k_w = 8$



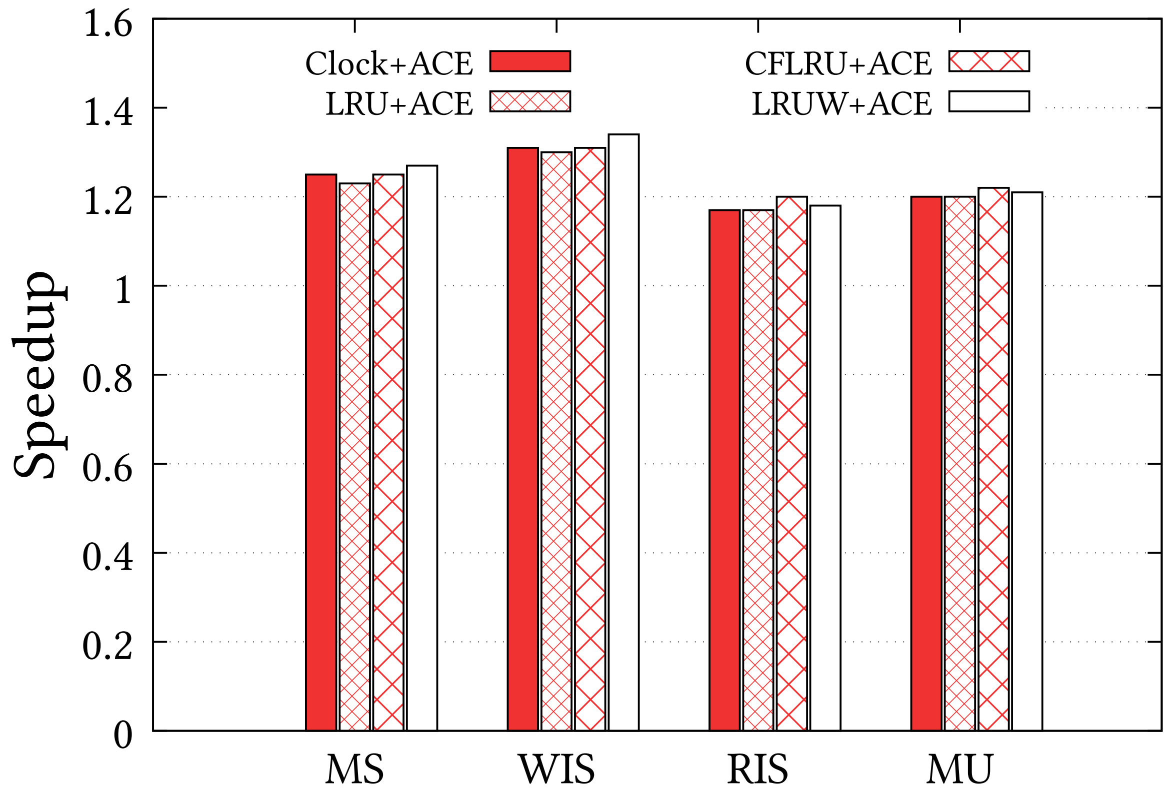
Even for devices with asymmetry 1.5, concurrency leads to up to 20% benefit

BOSTON UNIVERSITY

Experimental Evaluation

Device: Virtual SSD

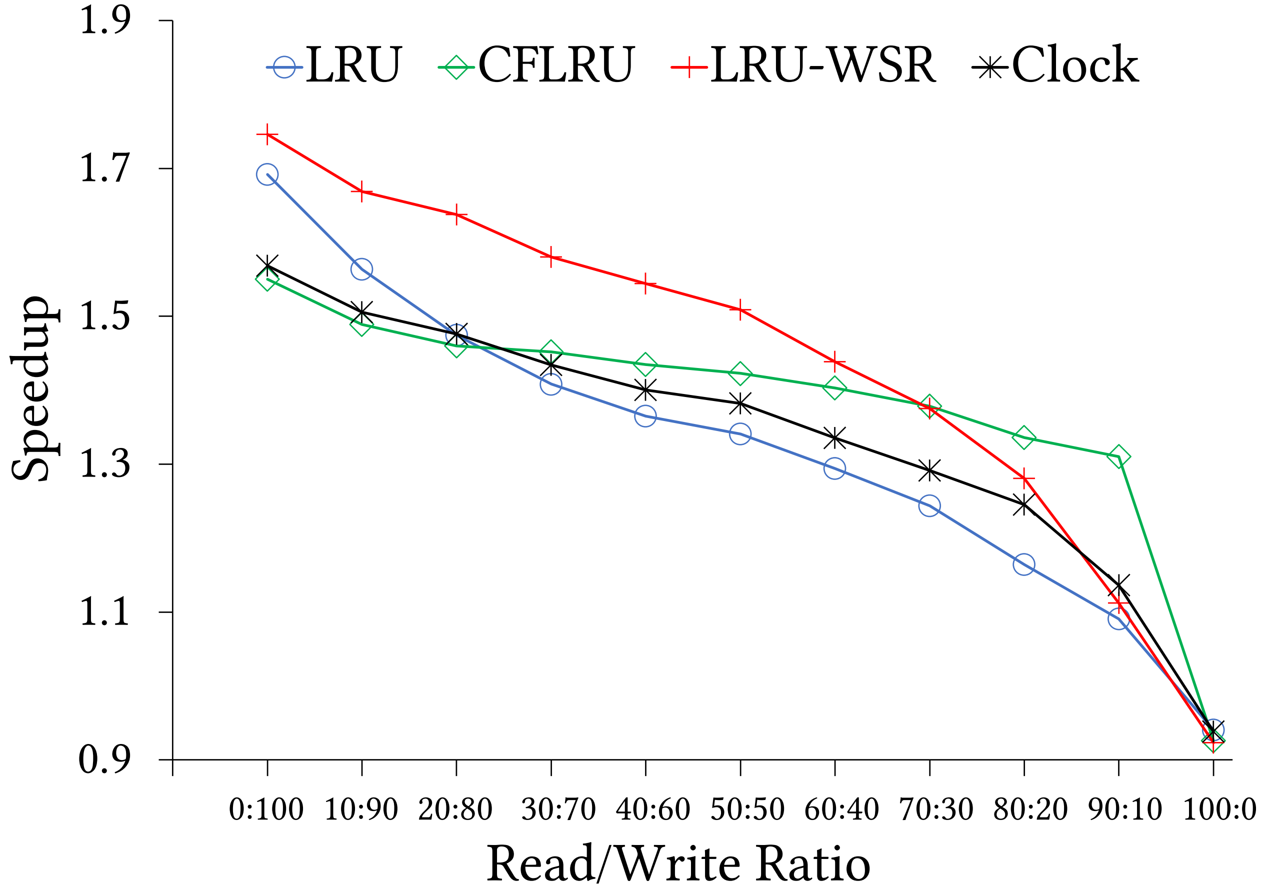
$\alpha = 2, k_w > 19$



The benefits remain for virtual SSDs that we cannot fully benchmark

BOSTON UNIVERSITY

Experimental Evaluation



Device: NVMe SSD

$\alpha = 3, k_w = 8$

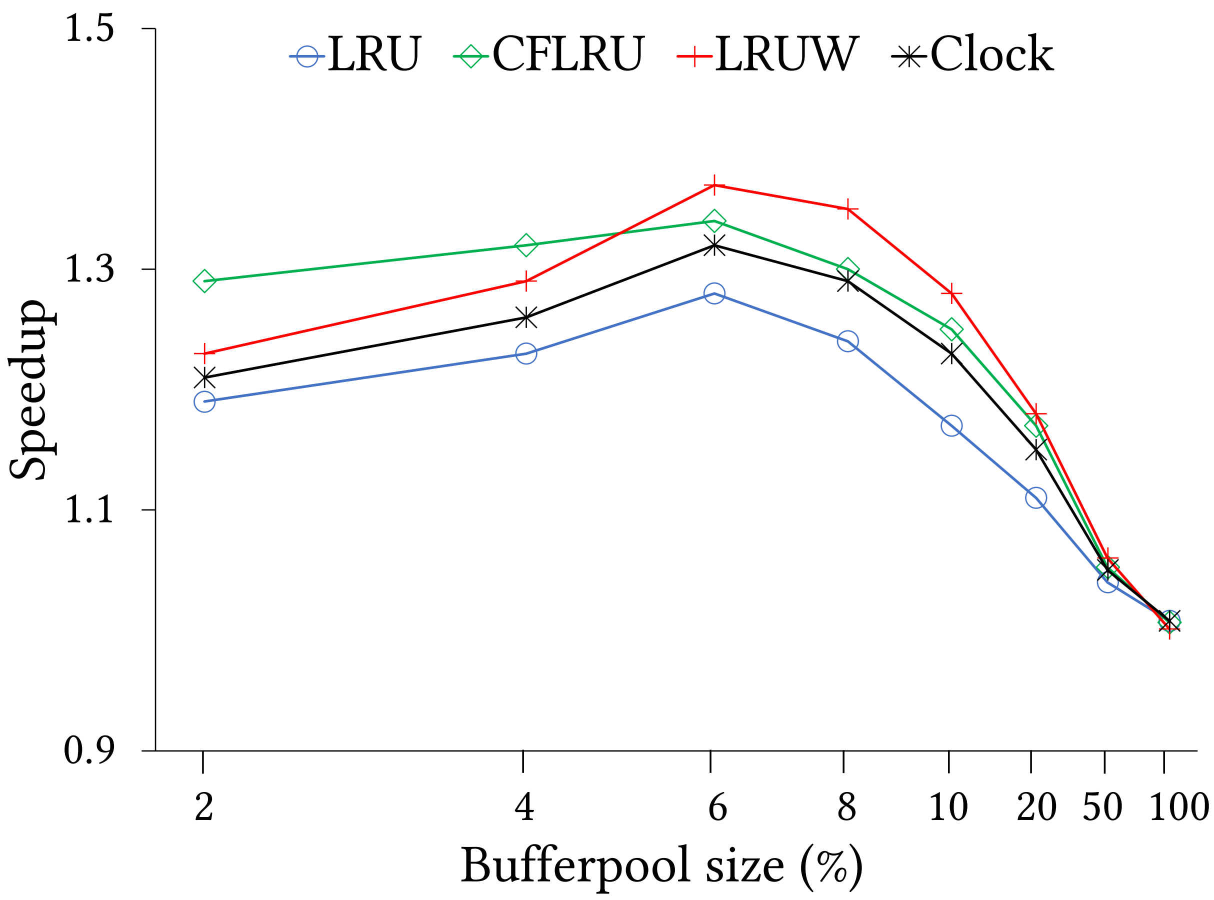
For write heavy workloads, gain of ACE can be as high as 1.75x

BOSTON UNIVERSITY

Experimental Evaluation

Device: NVMe SSD

$\alpha = 3$, $k_w = 8$

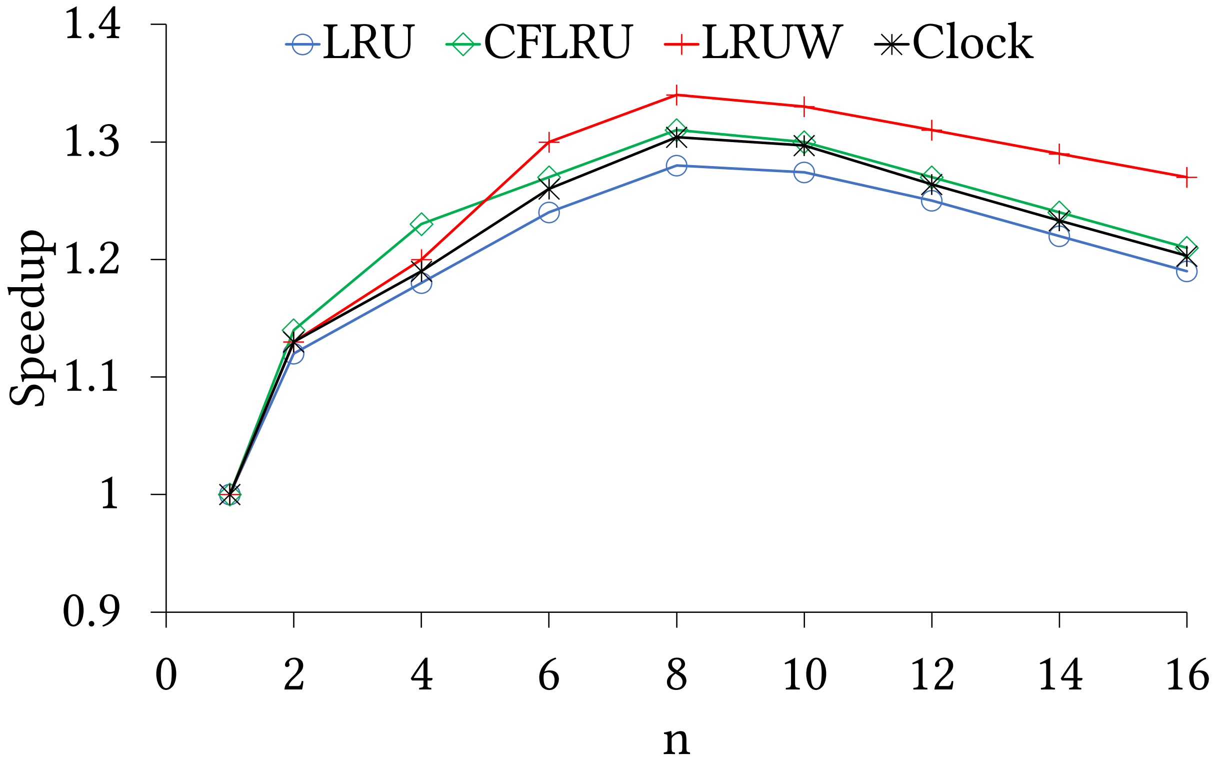


ACE performs particularly well under memory pressure

Experimental Evaluation

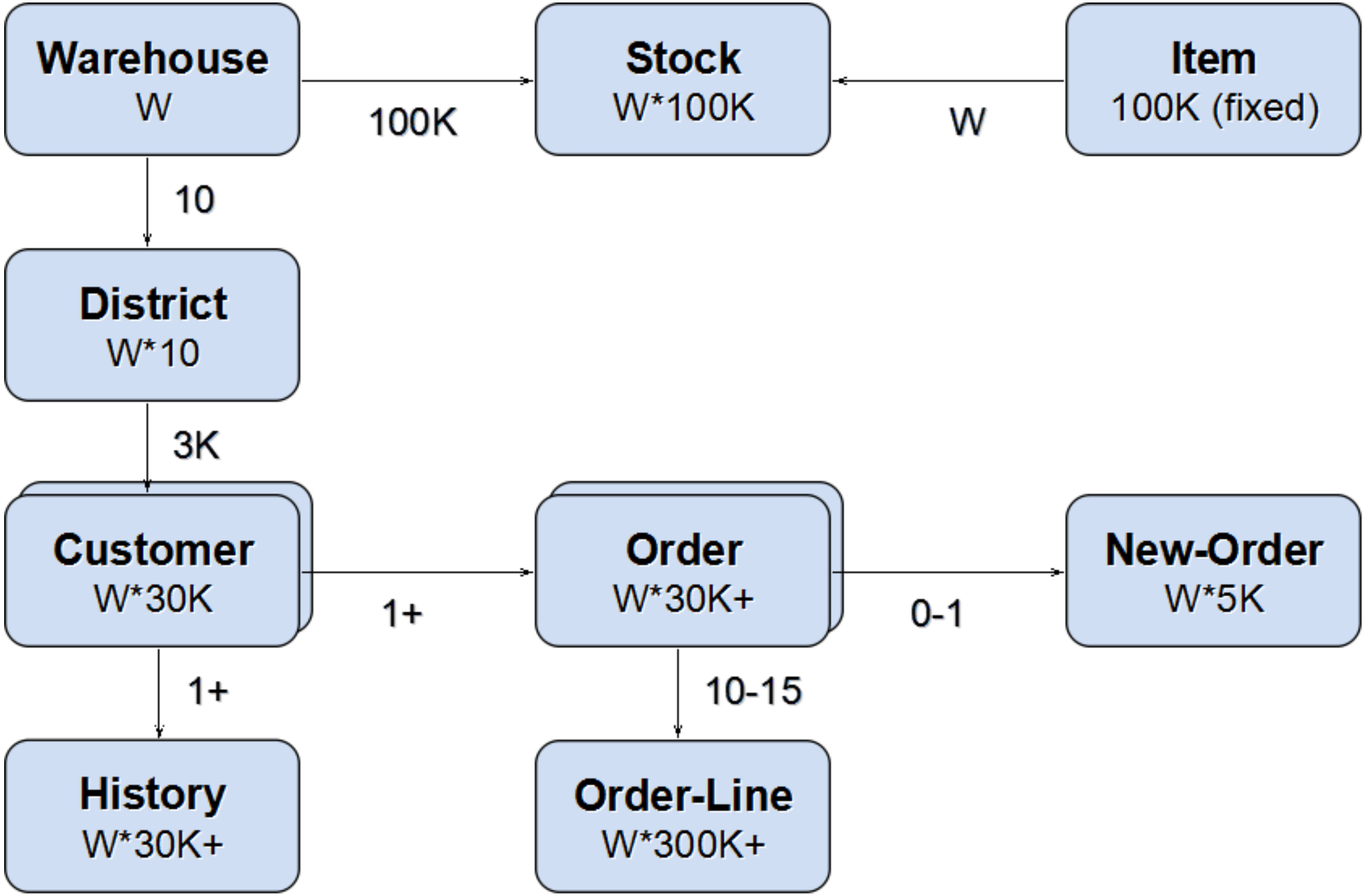
Device: NVMe SSD

$\alpha = 3, k_w = 8$



There is an optimal concurrency for each device.

Experimental Evaluation (TPC-C)



BOSTON UNIVERSITY

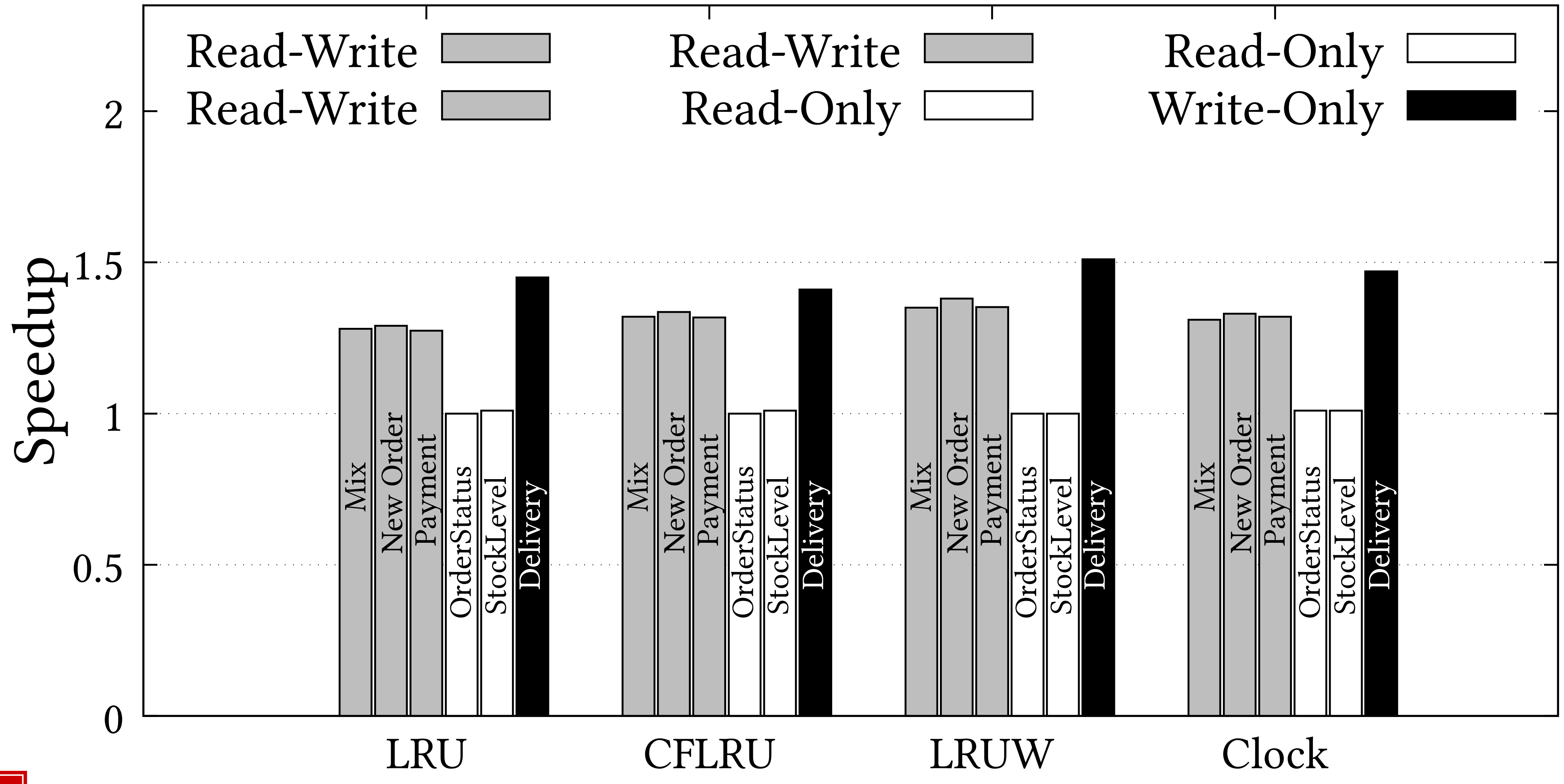
Experimental Evaluation (TPC-C)

TPC-C consists of 5 transactions

NewOrder (45%)	R/W Mix
Payment (43%)	R/W Mix
OrderStatus (4%)	R-only
StockLevel (4%)	R-only
Delivery (4%)	W-heavy

BOSTON UNIVERSITY

Experimental Evaluation (TPC-C)



BOSTON UNIVERSITY

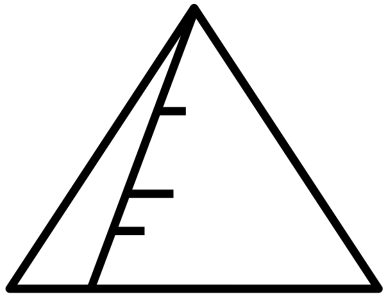
Conclusion

Make *asymmetry and concurrency* part of *algorithm design*

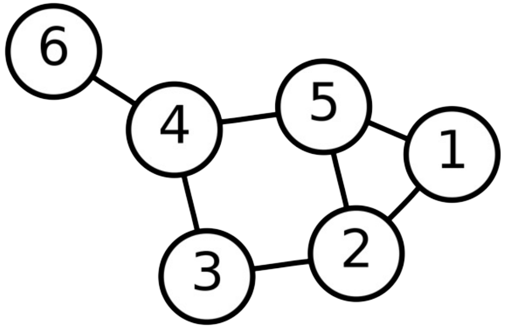
… not simply an engineering optimization

Build algorithms/data structures for storage devices
with asymmetry α and concurrency k

index structures



graph traversal algorithms



BOSTON UNIVERSITY

BOSTON UNIVERSITY

CS 561: Data Systems Architectures

class 18

Asymmetry & Concurrency Aware Storage Management

Prof. Manos Athanassoulis

https://bu-disc.github.io/CS561/